THÈSE DE DOCTORAT D'ÉTAT

ès Sciences Mathématiques

présentée

à l'Université Pierre et Marie Curie

Paris 6

par Jean-Louis LAURIERE

pour obtenir le grade de DOCTEUR ès SCIENCES.

Sujet de la thèse :

UN LANGAGE ET UN PROGRAMME POUR ENONCER ET RESOUDRE DES PROBLEMES COMBINATOIRES

soutenue le 21 Mai 1976

devant le jury composé de :

MM. J.ARSAC
C.F.PICARD
J.PITRAT
M.P.SCHÜTZENBERGER
L.SIKLÓSSY
J.C.SIMON

THÈSE DE DOCTORAT D'ÉTAT

ès Sciences Mathématiques

présentée

à l'Université Pierre et Marie Curie

Paris 6

par Jean-Louis LAURIERE

pour obtenir le grade de DOCTEUR ès SCIENCES.

Sujet de la thèse :

UN LANGAGE ET UN PROGRAMME POUR ENONCER ET RESOUDRE DES PROBLEMES COMBINATOIRES

soutenue le 21 Mai 1976

devant le jury composé de :

MM. J.ARSAC
C.F.PICARD
J.PITRAT
M.P.SCHÜTZENBERGER
L.SIKLÓSSY
J.C.SIMON

UNIV.
LEIDEN
BIBL.

Monsieur Jacques PITRAT m'a fait connaître un domaine de recherche passionnant, il m'a amicalement et constamment conseillé, guidé et encouragé au long de cette thèse; il anime une équipe dans laquelle chacun apprécie de pouvoir travailler et échanger des idées. Qu'il veuille bien trouver ici l'expression de toute ma reconnaissance et de ma profonde gratitude.

Je remercie Monsieur le Professeur J. ARSAC de m'avoir fait l'honneur d'accepter de présider le jury.

Je remercie Monsieur le Professeur M.P. SCHÜTZENBERGER qui a bien voulu s'intéresser à mon travail et faire partie du jury.

Je remercie également Messieurs C. PICARD, L. SIKLÓSSY et J.C. SIMON de leurs conseils et de leur participation au jury.

Je remercie le personnel de l'Institut de Programmation pour sa gentillesse et son efficacité, le laboratoire d'Informatique théorique et de Programmation de Paris 6 et Paris 7 qui m'a permis d'effectuer les passages. Je remercie enfin tout particulièrement Madame THUILLIER qui s'est chargée de la frappe de ce rapport.

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41548118_0023

L'informatique si elle se développe aujourd'hui avec et pour la gestion, l'économie, la médecine, les sciences humaines ou les mathématiques, marque un temps d'arrêt en tant que science propre.

La nouveauté fondamentale qu'apporte l'informatique n'a peut-être pas été, en effet, bien saisie. Les véritables problèmes qu'elle soulève, convergent vers ce qu'on nomme, d'un vocable peu heureux, l'"intelligence artificielle" ; c'est dans ce domaine que j'ai choisi de travailler.

Le but de l'intelligence artificielle est d'obtenir par programme un comportement intelligent et de résoudre des problèmes.

Il faut s'étonner de la résolution à grands frais, par l'informatique et la recherche opérationnelle de questions que nous abordons, nous, aisément à la main: le passage du bureau d'études, où des gens qui connaissaient à fond le problème oeuvraient avec papier et crayon, au système informatique a demandé, dans maintes entreprises à travers le monde, des années de travail à une équipe de mathématiciens et d'informaticiens.

Le chercheur opérationnel délaisse en effet trop souvent les méthodes et l'astuce utilisées manuellement, quelquefois sans même les étudier, pour un modèle

mathématique abstrait qui, tant dans sa formulation que dans sa résolution, s'éloignera radicalement du monde réel. Le prétexte est, en général, l'"impossibilité" de programmer les anciennes techniques humaines et la prétention d'atteindre un optimum. Cet optimum n'est que celui du modèle et quand bien même il est concrètement peu significatif, il satisfait l'esprit du mathématicien.

Ces pages décrivent une tentative de résolution par programme des problèmes de nature combinatoire. Utilisant la démarche générale et plusieurs techniques de l'intelligence artificielle, la généralité, la souplesse et l'efficacité sont à la fois recherchées.

Nous verrons que cette approche, qui est souvent fort semblable à celle de l'esprit humain, et s'adapte à chaque énoncé n'est pas contradictoire, dans les cas où cela a un sens précis, avec la recherche et la preuve de l'optimalité.

PRESENTATION GENERALE

L'analyse combinatoire, la théorie des graphes, la recherche opérationnelle fournissent tout un lot d'énoncés plaisants et délectables. S'y ajoutent les problèmes rencontrés dans le monde réel, en organisation, architecture, aide à la décision. Le schéma général du traitement informatique de tels énoncés peut être représenté ainsi :

	environnement concret	
étape A	↓	
	énoncé du problème	• *langage de problèmes*.
étape B	↓	
	algorithme de résolution	• *langage de description d'algorithmes*.
étape C	↓	
	programme	• *langage de programmation*.
étape D	↓	
	exécution en machine	• *instructions machine*.

La colonne de droite spécifie l'outil nécessaire à chaque étape. C'était habituellement à l'homme, à l'"analyste", qu'incombaient les points A et B. L'homme encore, le "programmeur", exécutait l'étape C. La machine n'intervenait que dans la phase ultime.

Au contraire, nous nous plaçons en A. Le problème étant énoncé dans le langage d'entrée, ALICE, c'est alors l'ordinateur qui se charge des étapes B, C et D.

METHODOLOGIE

Cinq caractéristiques méthodologiques originales doivent être soulignées.

• Formalisation des énoncés :

Les problèmes sont posés comme recherche de fonctions entre ensembles finis discrets. Le langage permet donc de manipuler les ensembles, les fonctions, les matrices, les opérateurs mathématiques usuels et les quantificateurs. La puissance

est celle d'une machine de Turing qu'il est d'ailleurs possible de simuler. La seule limitation qu'impose cette façon de faire est de ne pouvoir travailler dans le continu ou sur des ensembles infinis. L'énoncé seul du problème est demandé, aucun algorithme de résolution n'a besoin d'être décrit. Outre la souplesse et la concision qu'apporte le langage d'entrée dans le domaine combinatoire, l'énoncé formel définit avec rigueur le problème que l'utilisateur désire soumettre.

Cette façon de poser le problème en termes de fonctions renouvelle la programmation mathématique.

2• Généralité :

La procédure de résolution apparentée à la méthode "Branch and Bound" , utilise plusieurs principes de séparation et différents types d'énumération. Pour les problèmes de taille importante cette énumération est retardée le plus longtemps possible et le programme prévoit et élimine à l'avance la violation des contraintes du problème, à l'aide d'algorithmes d'implications (Chapitre II , paragraphes 4 et 6) et de préclusions (Chapitre III, paragraphe 1).

Le programme n'est pas une collection d'algorithmes particuliers mais bien une procédure unique. Cette procédure interprète l'énoncé et travaille sur trois représentations : un graphe , une représentation algébrique , des tests . Celles-ci permettent une représentation concise du problème, forment le support de raisonnements algorithmiques et elles coopèrent constamment.

3• Manipulation formelle :

Les contraintes algébriques forment un système dynamique. Il y a variation, en cours d'exécution, tant du nombre des contraintes que du nombre d'inconnues. L'algorithme d'unification et des algorithmes particuliers au domaine permettent, par étude formelle, et par échanges continuels avec la représentation graphique d'obtenir une bonne mémorisation et une gestion efficace de l'information contenue dans chaque contexte.

4• Réalisabilité et optimalité :

Le plan général du programme conduit celui-ci à juger de la difficulté *a priori* du problème. La distinction est nette entre obtention de solutions réalisables et optimalité. Dans une première phase, le but est d'exhiber une solution

quelconque, éventuellement triviale, satisfaisant toutes les contraintes. On s'attache, dans une seconde phase, à exhiber une bonne solution, dont la distance à l'optimum est mesurée : si la valeur de cette solution est Z_o et la borne inférieure de la fonction à optimiser est $\underline{Z}$, l'erreur relative par rapport à l'optimum Z^* est bornée

$$\frac{Z_o - Z^*}{Z^*} \leq \frac{Z_o - \underline{Z}}{\underline{Z}}$$ (cas d'une minimisation)

Le programme dispose alors, dans la troisième phase, de différentes techniques spécifiques pour atteindre et prouver l'optimalité.

5 • Adaptation du programme au problème posé.

Le programme construit dans chaque phase un plan de résolution par l'utilisation systématique et dirigée d'heuristiques. Pour un même énoncé formel, il fera, comme nous le ferions, un travail très différent si les données numériques sont différentes. Il joue constamment sur les coefficients, les dissymétries, la taille combinatoire ou toute particularité du problème proposé. La programmation automatique , adaptée à chacun des énoncés, est ainsi réalisée.

PLAN DE L'EXPOSE.

Un exemple complet de problème combinatoire est tout d'abord présenté. Sa formalisation dans le langage d'entrée ALICE ainsi que sa résolution sont alors explicitement exposées, de façon à donner dès le début au lecteur une idée des capacités du langage et du programme.

Les informations en ALICE sont fournies en deux temps : la définition du problème est donnée en notation mathématique; elle est suivie de données numériques éventuelles. Le chapitre I définit le langage, son dictionnaire, la description syntaxique et des exemples d'énoncés sont présentés.

L'interprétation et la mise en mémoire interne font ensuite l'objet du chapitre II . Les méthodes arborescentes de résolution, le plan et les heuristiques sont alors décrits au chapitre III .

Le chapitre IV réunit les résultats obtenus sur quelques 200 problèmes :

- analyse combinatoire,
- théorie des graphes,
- puzzles logiques ou arithmétiques,
- programmation linéaire en nombres entiers,
- partitionnement et recouvrement,
- programmation quadratique,
- voyageur de commerce, tournées de livraison, localisation,
- découpe linéaire,
- aide à la conception en architecture,
- coloriage et emplois du temps,
- cas concrets d'ordonnancements disjonctifs.

Le début du chapitre donne un tableau regroupant les caractéristiques des problèmes et les temps de calcul.

Le programme se montre compétitif avec des codes mis au point en recherche opérationnelle ou même plus performant que des programmes spécifiques d'un énoncé particulier.

Une table des symboles utilisés se trouve page 225, la table des matières est en page 226.

ENONCE ET RESOLUTION COMPLETE D'UN PROBLEME.

GENERATION DE PERMUTATIONS.

Ce problème m'a été proposé par M.P.Schützenberger.

Etant donné une permutation f sur les entiers de 1 à n , on définit le vecteur de monotonie m associé à f :

$$m(i) = 1 \Leftrightarrow f(i+1) > f(i) \qquad \forall\, i \in [1,n-1] \qquad (I)$$

On dit d'autre part que i , $i \in [1,n-1]$, est une avance dans f , si et seulement si l'entier f(i)+1 est placé, dans la permutation, à droite de f(i). Cela est équivalent à :

i n'est pas une avance $\Rightarrow$

$$\forall\, j \in [i+1,n] \qquad f(j) \neq f(i)+1$$

i est une avance $\Rightarrow$

$$\forall\, j \in [1,i-1] \qquad f(j) \neq f(i)+1 \quad \text{et} \quad f(i) \neq n\ .$$

Ainsi, dans la permutation

4 5 8 1 9 2 6 3 7

le vecteur des monotonies est

m = (1 1 0 1 0 1 0 1 0)

l'ensemble des avances est {1,2,3,4,6,7} et le vecteur caractéristique de cet ensemble

a = (1 1 1 1 0 1 1 0 0)

Le problème est d'énumérer toutes les permutations d'ordre donné et qui admettent un vecteur d'avances et un vecteur de monotonies donnés.

Une définition mathématique de ce problème, à la fois concise et exhaustive,

telle qu'il serait agréable de la soumettre à une machine, est de la forme :

Soit une constante n ,
Soit l'ensemble X1 des entiers de 1 à n ,
Soit l'ensemble X2 des entiers de 1 à n-1 ,
Soient deux coefficients m et a d'entiers sur X1 ,
Trouver une bijection f de X1 sur lui-même avec

$\forall\, i \in X2 \quad m(i)=1 \Leftrightarrow f(i+1) > f(i)$ *(I)*

$a(i)=1 \Rightarrow \forall\, j \in [1,i-1], f(j)\neq f(i)+1 \text{ et } f(i)\neq n$ *(II)*

$a(i)=0 \Rightarrow \forall\, j \in [i+1,n], f(j)\neq f(i)+1$ *(III)*

à la suite de cela, les données numériques n , m et a seraient lues.

Nous verrons que ceci est possible dans le langage ALICE et est présenté exactement sous la forme :

```
SCI CST N
    ENS X1  =   INT 1   N
        X2  =   INT 1   -   N   1
    CCE M   SUR X1
    CCE A   SUR X1
IRC BIJ F   >-> X1  X1
AVE CCS APP I   X2  <=> =   M   I   1   >   F   +   I   1   F   I
    CCS APP I   X2  ==> =   A   I   1   ET  QQS APP J   INT 1   -   I   1   ¬=
F   J   +   F   I   1   ¬=  F   I   N
AVE CCS APP I   X2  ==> =   A   I   C   QQS APP J   INT +   1   I   N   ¬=  F
J   +   F   I   1

FIN
9,
1,1,C,1,C,1,C,1,C,
1,1,1,1,C,1,1,C,0,
FIN
```

RESOLUTION

Lors de l'interprétation de l'énoncé précédent le programme construit en particulier le graphe :

où tout sommet à gauche est a priori relié par un arc à chaque sommet de droite.

L'expression (I) donne naissance aux huit contraintes (normalisées) :

(1) $0 \leq -1 - f1 + f2$

(2) $0 \leq -1 - f2 + f3$

(3) $0 \leq -1 + f3 - f4$

(4) $0 \leq -1 - f4 + f5$

(5) $0 \leq -1 + f5 - f6$

(6) $0 \leq -1 - f6 + f7$

(7) $0 \leq -1 + f7 - f8$

(8) $0 \leq -1 - f8 + f9$

En effet, quand le programme rencontre un indice i pour lequel $m(i) \neq 1$, il prend :

$$\neg \left[f(i+1) > f(i) \right] \text{ soit}$$
$$f(i+1) \leq f(i)$$

ou bien encore, en se servant du fait que f est une injection :

$$f(i+1) \leq f(i) - 1 .$$

Les expressions (II) et (III) donnent :

(9) $0 \neq -1 + f1 - f2$

(10) $0 \neq -1 + f1 - f3$

(11) $0 \neq -1 + f2 - f3$

(12) $0 \neq -1 + f1 - f4$

(13) $0 \neq -1 + f2 - f4$

(14) $0 \neq -1 + f3 - f4$

(15) $0 \neq -1 + f1 - f6$

(16) $0 \neq -1 + f2 - f6$

(17) $0 \neq -1 + f3 - f6$

(18) $0 \neq -1 + f4 - f6$

(19) $0 \neq -1 + f5 - f6$

(20) $0 \neq -1 + f1 - f7$

(21) $0 \neq -1 + f2 - f7$

(22) $0 \neq -1 + f3 - f7$

(23) $0 \neq -1 + f4 - f7$

(24) $0 \neq -1 + f5 - f7$

(25) $0 \neq -1 + f6 - f7$

(26) $0 \neq +1 + f5 - f6$

(27) $0 \neq +1 + f5 - f7$

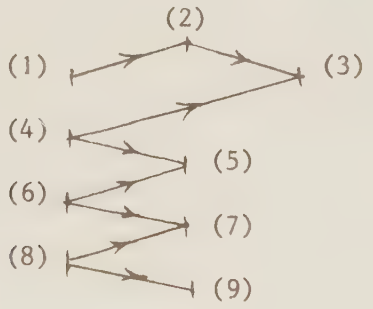
(28) $0 \neq +1 + f5 - f8$

(29) $0 \neq +1 + f5 - f9$

(30) $0 \neq +1 + f8 - f9$

et de plus par (II) , les arcs joignant les sommets de départ 1, 2, 3, 4, 6 et 7 à l'image 9 ont été supprimés.

Les contraintes intéressantes sont celles en $\leq$, à partir desquelles une fonction de Grundy sur les inconnues associée au schéma :



est construite (fonction $\mathcal{G}$, chapitre II§3) .

i	1	2	3	4	5	6	7	8	9
$\mathcal{G}$(i)	1	2	3	1	2	1	2	1	2

La contrainte (1) est donc (ChII§5) examinée en premier , elle entraine (ChII§4):

$0 \leq -1 + 7$ qui est trivialement vérifiée

$0 \leq -f1 + 7$ qui conduit à supprimer $\overrightarrow{1.8}$

$0 \leq +f2 - 2$ qui conduit à supprimer $\overrightarrow{2.1}$

Les sept autres contraintes sont considérées dans l'ordre induit par $\mathcal{G}$, qui se trouve être ici l'ordre naturel. Elles mènent à supprimer en outre les arcs $\overrightarrow{2.8}$, $\overrightarrow{3.1}$ et $\overrightarrow{3.2}$, $\overrightarrow{4.8}$, $\overrightarrow{5.1}$, $\overrightarrow{6.8}$, $\overrightarrow{7.1}$, $\overrightarrow{8.8}$ et $\overrightarrow{8.9}$ et enfin $\overrightarrow{9.1}$.

Il n'y a pas d'autres implications à ce stade puisque l'on ne peut rien déduire ici des inéquations (9) à (30) et que le graphe n'engendre aucune conclusion. Il n'y a aucune élimination d'inconnues par contraintes en égalité.

Etape de choix :

Le sommet arrivée 9 ne peut être atteint que par les points 5 et 9 . C'est sur ces antécédents que se fait le choix puisque tous les sommets de départ ont au moins trois images possibles et que l'on cherche une surjection. Le programme fait le

choix $f(5) = 9$.

Après suppression des arcs $\overrightarrow{5.2}$, $\overrightarrow{5.3}$, $\overrightarrow{5.4}$, $\overrightarrow{5.5}$, $\overrightarrow{5.6}$, $\overrightarrow{5.7}$, $\overrightarrow{5.8}$ et $\overrightarrow{9.9}$ la nouvelle règle "f(5) est réécrit 9" donne :

(4),(5),(19),(26),(27),(28),(29) trivialement vraies ;

(24) implique $f(7) \neq 8$ et un nouvel examen

de (6) et (9) $\Rightarrow$ $f(6) \neq 7$,
$f(8) \neq 7$,

et c'est de nouveau la fin des implications.

Le prochain choix est $f(3)=8$, et la première solution atteinte :

i	1	2	3	4	5	6	7	8	9
f(i)	6	7	8	1	9	3	4	2	5

Cinq choix seulement ont été faits alors, le programme jugeant qu'il reste assez peu d'arcs dans le graphe et assez de contraintes en $\neq$ pour lancer une énumération aveugle qui produit donc, coup sur coup, les variantes :

6 7 8 1 9 2 4 3 5
6 7 8 2 9 3 4 1 5
6 7 8 3 9 1 4 2 5

Les choix successifs sont effacés et de nouvelles solutions obtenues. La remontée dans l'arborescence s'effectue jusqu'à la négation du premier choix. On a alors l'implication f(9)=9 , de nouveaux choix sont effectués jusqu'à ce que la pile des choix soit vide.

Après 217 retours en arrière le programme obtient les 42 solutions du problème. Il faut en moyenne 0.5 seconde (IBM 370-168) par solution.

LE LANGAGE

.1. LES PROGRAMMES EXISTANTS :

Un grand nombre de programmes ont été écrits pour résoudre des problèmes combinatoires, c'est-à-dire des problèmes où n'interviennent que des ensembles discrets.

Ces programmes résolvent un type de problème bien déterminé : citons les programmes de programmation linéaire en nombre entiers : Gilmore et Gomory (1961) (1963) , Balinski (1965) (1967) et (1970) , Balas (1965) et (1967) , Gondran (1974) , les programmes utilisés dans les compagnies aériennes, depuis 1965 aux USA , pour la fabrication automatique des rotations d'équipages, les programmes de résolution du célèbre problème du voyageur de commerce : Little et al (1963) , Held et Karp (1971) , les problèmes d'emplois du temps Lions (1967), de Werra (1970) , les problèmes de collectes ou de distributions Balinski et Quandt (1964) , Lemaire (1972) , les problèmes non linéaires en variables booléennes Hammer et Rudeanu (1968) , Hansen (1974).

En outre, cinq programmes utilisent des méthodes d'intelligence artificielle pour aborder des problèmes combinatoires :

Tonge (1965) , Fikes (1968) , Pugh (1972) , Pfefferkorn (1973) et Laurière (1971).

Le langage LANG-SUMT de Robert E. PUGH (1972) :

Ce langage et son interpréteur ont été écrits pour faciliter l'emploi du programme général SUMT de Fiacco et Mac Cormick (1964) en programmation non linéaire. Les valeurs numériques des fonctions, de leurs gradients et des dérivées partielles secondes sont obtenues après un calcul formel à partir de la représentation algébrique de ces fonctions. L'efficacité à l'exécution est atteinte par une forme canonique "factorisée" des expressions.

L'écriture de sous-programmes spéciaux pour chaque dérivée et chaque problème, avec les erreurs humaines possibles, est ainsi évitée au programmeur. Cette approche, qui permet une optimisation du code, devrait en outre, améliorer finalement le temps de calcul total.

Le système REF-ARF de Richard FIKES (1968) :

Fikes a défini un langage REF dans lequel il est possible d'écrire des procédures non déterministes, c'est-à-dire dans lesquelles des choix restent à faire au moment de l'exécution, pour résoudre des puzzles combinatoires. L'interpréteur ARF dirige alors l'exécution de façon à parvenir à l'instruction fin et donc à une solution au plus vite.

L'interpréteur manipule formellement les expressions et retarde les choix le plus longtemps possible.

```
    begin;                                                                    1
       set vector A1 to X,S,E,N,D;                                            2
       set vector A2 to X,M,O,R,E;                                            3
       set vector SUM to M,O,N,E,Y;                                           4
       set vector L to D,N,E,S,R,O,M,Y;                                       5
       for I → 8 do to L1;                                                    6
L1:    set <L[<I>]> to select(0,9);                                           7
       condition excl(<L[1]>,<L[2]>,<L[3]>,<L[4]>,<L[5]>,<L[6]>,              8
          <L[7]>,<L[8]>);                                                     9
       condition ~(<M> = 0)∧~(<S> = 0);                                       10
       set <CARRY> to 0;                                                      11
       for J → 4 do to L2;                                                    12
       set <I> to 6 + -<J>;                                                   13
       if <A1[<I>]> + <A2[<I>]> + <CARRY> < 10 then L3;                       14
       condition <A1[<I>]> + <A2[<I>]> + <CARRY> = 10 + <SUM[<I>]>;           15
       set <CARRY> to 1;                                                      16
       goto L2;                                                               17
L3:    condition <A1[<I>]> + <A2[<I>]> + <CARRY> = <SUM[<I>]>;                18
       set <CARRY> to 0;                                                      19
L2:;                                                                          20
       condition <M> = <CARRY>;                                               21
    end;                                                                      22
```

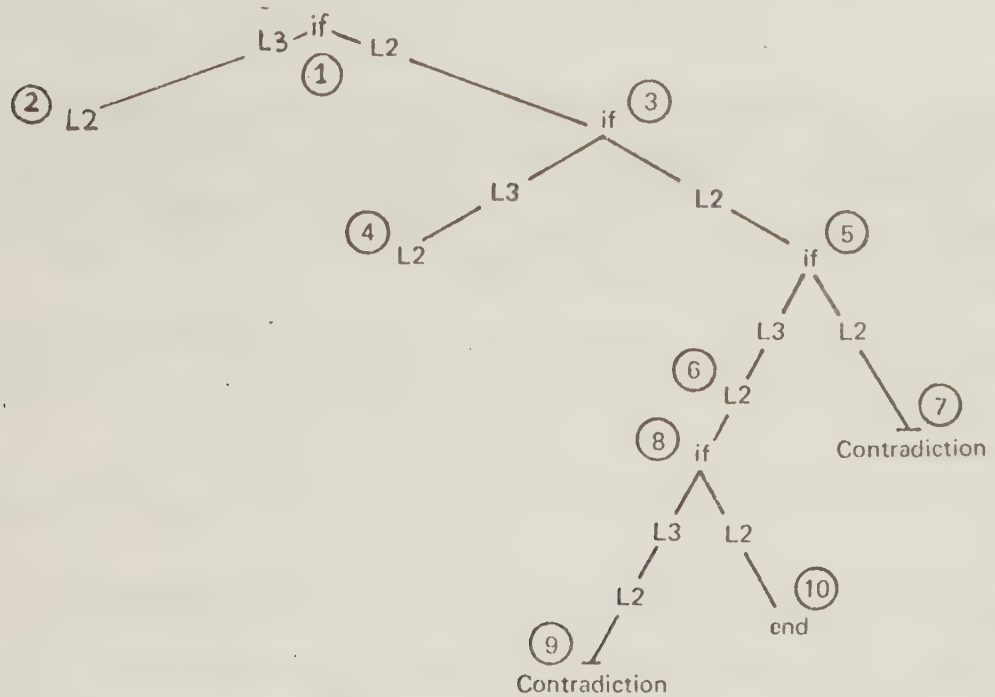
Procédure REF pour résoudre le problème SEND+MORE = MONEY , dans lequel il faut faire correspondre à chaque lettre un chiffre différent de sorte que l'addition dans le système décimal soit correcte.

L'instruction x:=select (i,j) signifie que les valeurs de x doivent être prises dans l'intervalle des entiers de i à j .

exclude $(y_1,y_2,\ldots,y_n)$ signifie que les valeurs des y_i doivent toutes être différentes.

Les idées de base de ARF paraissent excellentes. Le fait qu'il faille donner en entrée une procédure pour résoudre le problème posé est cependant un handicap.

D'ailleurs, l'existence de if et de goto dans REF oblige ARF à créer autant de contextes non nécessaires à la résolution, qui obligent à des retours en arrière même quand cela n'est pas nécessaire. Exemple :



Arborescence de recherche par ARF pour SEND+MORE = MONEY.

[Les retours en arrière, correspondant aux hypothèses sur les retenues ne sont d'aucune utilité. Notre programme trouve ici la solution sans arborescence (cf. partie IV).]

Un reproche grave est aussi celui que H.Siklossy (1971)fait aux programmes de démonstrations de théorèmes, à savoir qu'il est inutile d'écrire des systèmes sophistiqués pour résoudre des problèmes où la méthode combinatoire (exploration explicite) est satisfaisante :

Il faut à ARF 8 minutes d'IBM 360/67 et l'interprétation de quelques 700 instructions pour résoudre le problème des "missionnaires et cannibales" (2'27 pour SEND+MORE=MONEY).

Le traitement de problèmes de dimensions respectables nécessitait de meilleures performances.

Le traitement de problèmes d'optimisation exigeait de nouvelles méthodes, une plus grande puissance et une plus grande souplesse pour énoncer les problèmes, l'usage de quantificateurs et la paramétrisation, la présence d'objets mathématiques plus riches et enfin la mise en oeuvre d'heuristiques plus fines.

Partant des idées et des problèmes posés par Fikes, c'est ce vers quoi tend le système qui va maintenant être décrit.

I.2. LE LANGAGE.

Le langage d'entrée ALICE* a été conçu pour poser des problèmes. L'utilisation d'ensembles , d'opérations sur les ensembles, de produits cartésiens d'ensembles, de vecteurs, de matrices, de chemins, d'arbres, de circuits, de bijections, est sa première caractéristique.

Il permet d'énoncer toute contrainte utilisant des symboles logiques ou algébriques parmi lesquels :

+ , * , = , ≥ , Σ , mod (modulo) , ∃ , ∀ , ∈ , ⇒ , ⇔ , ou , et , ¬ (non logique) etc ... .

L'énoncé d'un problème se décompose de façon générale en quatre parties :

a. une partie déclarative commandée par des "soit" .
b. une partie indiquant ce que l'on cherche.
c. une partie exprimant les contraintes.
d. les données numériques.

a) Partie déclarative.

Comme dans tout langage de programmation, chaque symbole utilisé dans une instruction doit avoir été précédemment déclaré explicitement ou implicitement.

Exemples :

soi cst N (1)

déclare N comme constante .

soi ens E = int 1 N (2)

définit E comme l'ensemble des entiers de 1 à N .

soi coe C sur E (3)

définit C comme coefficient numérique sur l'ensemble E .

---

* A Langage for Intelligent Combinatorial Exploration.

Dans les cas (2) et (3) des données numériques suivront l'énoncé formel du problème, qui donneront des valeurs à la constante N et aux coefficients C_i pour i=1,...,N.

b) Objets cherchés

Dans tous les problèmes, on cherchera à attribuer une valeur à une (des) constante(s) inconnue(s) ou/et à construire une (des) application(s) d'un ensemble fini dans un ensemble fini.

Ainsi avec "tro" pour trouver :

tro cst M

trouver la valeur de constante M .

tro fon G >-> E F

trouver une fonction G de E vers F .

c) Contraintes

Les contraintes sont précédées de l'abréviation AVE (avec) et sont exprimées à l'aide des symboles mathématiques usuels et de mots clés spécifiques au langage (voir dictionnaire d'ALICE, tableau I.3) . Les expressions sont en polonaise préfixée. Cette notation linéaire et non ambiguë a en effet été préférée à la notation habituelle.

Exemples : AVE MIN M

avec M minimum

AVE ≤ 1 + X 1 X 2

avec $x_1 + x_2 \geq 1$

AVE MAX + X 1 - * 3 * X 2 X 3 X 4

avec $x_1 + 3x_2 . x_3 - x_4$ maximum

AVE > SIG APP I E G I M

(Σ) (ϵ)

(où E est un ensemble précédemment déclaré)

avec $\sum_{i \epsilon E} G(i) > M$.

A la suite des contraintes le mot clé "FIN" spécifie la fin des données formelles et le début des données numériques.

Les données numériques sont alors lues par le programme dans l'ordre où elles ont été déclarées dans la partie a).

⟨instruction⟩ := ⟨déclaration⟩ | ⟨condition⟩ | FIN

⟨déclaration⟩ := SOI CST ⟨nom⟩ | SOI ENS ⟨nom ens.⟩ = ⟨déf.ens.⟩ |
SOI ⟨vecteur⟩ ⟨nom⟩ SUR ⟨nom ens.⟩ | TRO CST ⟨nom⟩ |
TRO ⟨qualificateur⟩ ⟨nom fon.⟩ → ⟨nom ens.⟩ ⟨nom ens.⟩ ⟨option⟩

⟨condition⟩ := AVE ⟨exp.logique⟩ | AVE ⟨exp.algébrique⟩

⟨exp.logique⟩ := ⟨op.logique⟩ ⟨exp.logique⟩⟨exp.logique⟩ |
⟨op.algébrique⟩ ⟨exp.algébrique⟩ ⟨exp.algébrique⟩ |
⟨quantificateur⟩ APP ⟨indice⟩ ⟨nom ens.⟩ ⟨exp.logique⟩ |
¬ ⟨exp.logique⟩

⟨exp.algébrique⟩ := ⟨op2⟩ ⟨exp.algébrique⟩ ⟨exp.algébrique⟩ |
⟨op1⟩ ⟨exp.algébrique⟩ | ⟨nom fon.⟩ ⟨exp.algébrique⟩ |
SIG APP ⟨indice⟩ ⟨nom ens.⟩ ⟨exp.algébrique⟩ |
MAT ⟨exp.algébrique⟩ ⟨exp.algébrique⟩ |
⟨vecteur⟩ ⟨exp.algébrique⟩ | ⟨entier⟩

⟨qualificateur⟩ := FON | FMU | BIJ | INJ | SUJ | RES

⟨op1⟩ := MNS | MIN | MAX

⟨op2⟩ := + | * | / | - | MOD

⟨op.logique⟩ := OU | ET | ⇒ | ⇔

⟨op.algébrique⟩ := < | <= | > | >= | = | ¬=

⟨quantificateur⟩ := QQS | EXI

⟨vecteur⟩ := COE | COA

⟨déf.ens.⟩ := INT ⟨cste⟩ ⟨cste⟩ | OBJ | PDT ⟨nom ens.⟩ ⟨nom ens.⟩

⟨indice⟩ := I | J | K | L | ⟨entier⟩

⟨option⟩ := ⟨type option⟩ ⟨option⟩ | ⟨type option⟩

⟨type option⟩ := SUC | DIS | CIR | ARB | DMI | DMA | MAX | VAL | SYM

⟨nom⟩ := ⟨lettre⟩ ⟨nom⟩ | ⟨lettre ⟩

⟨entier⟩ := ⟨chiffre⟩ ⟨entier⟩ | ⟨chiffre ⟩

I.3 - PRECISIONS D'UTILISATION DU LANGAGE ET DICTIONNAIRE.

Les mots sont liés sur 3 caractères calés à gauche et séparés par un blanc à raison de 20 mots par ligne.

Les mots du dictionnaire sont réservés.

Tout mot placé en tête de ligne est différent des quatre mots clés : SOI, TRO, AVE et FIN est considéré avec la suite de la ligne comme un commentaire.

Dictionnaire du langage ALICE

L'avant-dernière colonne donne l'ordre du symbole en notation polonaise (nombre de termes commandés par ce symbole).

Une étoile en dernière colonne indique que le symbole provoque la lecture de données numériques qui lui sont associées.

Symbole	signification	syntaxe (exemple)	Ordre	Lecture
SOI	soit		1	
TRO	trouver	suivi de CST ou ⟨ fonction ⟩	1	
AVE	avec		1	
FIN	fin		0	
CST	constante	suivi d'un ou plusieurs nom(s)	-	(*)
ENS	ensemble	suivi d'un nom	1	
COE	vecteur de coefficients numériques	"	1	*
COA	vecteur de coeff. alphanumériques	"	1	*
MAT	matrice (numérique)	"	2	*
FON	fonction univoque	"	1	
FMU	fonction multivoque	"	1	
INJ	injection	"	1	
SUJ	surjection	"	1	
BIJ	bijection	"	1	
RES	restriction	"	1	
SUC	successeurs imposés (graphe)	caractéristiques d'une fonction	0	*
SYM	symétrie des successeurs			

VAL	valuations des arcs	"	0	*
DIS	disjonction imposées	"	0	*
DMA	degré maximum des images	"	0	*
DMI	degré minimum des images	"	0	*
CHE	chemin	"	2	
CIR	circuit	"	0	
ARB	arbre	"	0	
SUS	ensemble des successeurs d'un point	"	1	
PRE	" des prédécesseurs	"	1	
END	fin de l'exécution.			
SUR	sur	COE SUR E	1	
PDT	produit cartésien	PDT E F	2	
INC	inclus dans		1	
APP	appartenant à (ϵ)		1	
>->	de ... vers ...	G >-> E F	2	
INT	intervalle entier fermé	INT 4 9	2	
OBJ	ensemble d'objet	E = OBJ A B C ...	-	
QQS	quel que soit.(∀)		2	
EXI	il existe au moins un.(∃)		2	
¬	négation logique		1	
ET	et "		2	
OU	ou "		2	
==	entraine		2	
⇔	est équivalent à		2	
--	se réécrit		2	
SIG	sigma		2	
MIN	minimum		1	
MAX	maximum		1	
+	+		2	
-	moins binaire		2	
MNS	moins unaire		1	
*	*		2	
/	division		2	
MOD	modulo		2	

<	<		2
<=	≤		2
>	>		2
>=	≥		2
=	=		2
¬=	≠		2
I ,J ,K ,L ,II ,JJ ,KK ,LL		indices réservés	1
1 ,...,999		constantes numériques	1

L'utilisation de certains mots-clés spécifiques du langage et non encore rencontrés comme OBJ ou END sera décrite plus loin dans les résultats.

Tout symbole n'appartenant pas au dictionnaire est considéré comme le nom d'un nouvel objet qui doit être déclaré dans une instruction "SOI".

Lecture des données numériques :

Neuf mots clés provoquent la lecture dans l'ordre où ils apparaissent dans l'énoncé, de données associées.

Pour CST , COE , COA , MAT , VAL , DMI , DMA le programme sait à l'avance le nombre de données à lire : il suffit de les mettre les unes derrière les autres séparées par des virgules, à raison d'un nombre quelconque par carte.

Pour DIS (resp. SUC) les sommets en disjonction (respectivement successeurs) pour un sommet donné sont lus séparés par des virgules à raison de 20 par cartes; la dernière carte concernant un sommet étant systématiquement complétée par des zéros séparés par des virgules.

Après avoir lu un énoncé le programme le met en mémoire et tente de résoudre le problème qui lui a été posé.

La mise en mémoire de l'information et son traitement feront l'objet du deuxième chapitre. Mais, d'ores et déjà, les différents types de contraintes et

d'optimisation qui sont couramment rencontrés en analyse combinatoire ou en recherche opérationnelle sont présentés dans les deux paragraphes suivants.

I.4 - LES CONTRAINTES.

Les principales contraintes apparaissant, tant dans des problèmes purement combinatoires, qu'en recherche opérationnelle (optimisation), peuvent être classées en trois grands types.

Dans les deux premiers types la contrainte s'exprime aisément et explicitement en fonction des inconnues. Le premier type regroupe alors les cas les plus simples des contraintes qui ne font intervenir que deux termes exactement; le second type concernant les autres cas.

Exemples

1er type

a) $x_1 = 3$
b) $x_1 \neq 3$
c) $x_1 = x_2$
d) $x_1 \neq x_2$

2ème type

a) $x_1 = x_2 + 3$
b) $4x_1 - 3x_3 - 8 \geq 0$
c) $|x_1 - x_2| \geq 4$ soit $x_1 - x_2 \geq 4$ OU $x_2 - x_1 \geq 4$.
d) $x_1 = 3$ OU $x_1 = x_2$.

Le troisième type correspond aux autres contraintes, notamment à toutes celles qui ne s'expriment pas explicitement en fonction d'inconnues, ainsi :

3ème type

a) Trouver un graphe partiel d'un graphe donné, qui soit un arbre.

b) $\sum_{i \mid \mathcal{R}(i)} \text{coef}(i) \leq \alpha$ (α constante dépendant de $\mathcal{R}(i)$)

où $\mathcal{R}(i)$ est une certaine propriété dépendant des valeurs données aux

inconnues x_i .

Ce dernier cas est très couramment rencontré : Problèmes de découpe, de tournées etc...

Découpe

$\mathcal{R}(i)$ s'écrit alors: $f(i) = j$ (j constante donnée),

et b) : $\sum_{\{i \mid f(i)=j\}} \text{coef}(i) \leq \alpha_j$.

La somme des longueurs des pièces découpées dans le même morceau j (pièces i telles que $f(i)=j$) doit être inférieure à la taille de celui-ci.

Tournées

$\mathcal{R}(i)$ est l'appartenance à un chemin.

La somme des distances parcourues par un camion doit être inférieure à l'autonomie de celui-ci.

$$f(i) = j \Rightarrow \mathcal{R}(i,j)$$

Il est clair que dans ce troisième type, dès que l'on sait énoncer mathématiquement la propriété $\mathcal{R}$, on peut (théoriquement) écrire celle-ci en fonction d'inconnues le plus souvent booléennes.

Mais cette façon de faire peut conduire à

- augmenter considérablement le nombre des inconnues; ainsi en b) , il faut poser

$$\forall i, \forall j, \; y_{ij} = \begin{cases} 1 & \text{si l'image de } i \text{ est } j \\ 0 & \text{sinon} \end{cases}$$

- augmenter le nombre de contraintes; il faut ainsi n contraintes additionnelles, dans un graphe à n sommets, pour traduire l'arboricité.

- donner tout simplement un modèle inexploitable pour un problème simple. En effet bien que l'on puisse toujours, mais souvent à grand renfort de nouvelles inconnues "artificielles" , se ramener au cas linéaire, la méthode simpliciale est d'un maigre secours quand on a des inconnues booléennes.

Aussi après des modèles prometteurs, tels GOTLIEB (1963) sur les problèmes d'emploi du temps ou BESSIERE (1965) , sur la recherche du nombre chromatique d'un graphe par un programme linéaire en nombres entiers, il faut maintenant reconnaître que le fait d'introduire une nouvelle inconnue à tout propos ne saurait constituer une solution dans une science qui se veut opérationnelle. Ces introductions de variables auxiliaires sont ici évitées.

A ces différents type de contraintes correspondent, dans le programme, autant d'actions différentes qui seront décrites aux paragraphes II.1, II.2 et II.3.

Les premières seront toujours exploitées a priori puis détruites. Le programme ajoutera à cet ensemble de contraintes simples celles de la forme

$$|\{i|f(i) = j\}| \gtrless \alpha$$

Les secondes seront étudiées à chaque itération, selon un algorithme figurant en II.2.2. ; elles deviendront d'ailleurs du premier type lorsque suffisamment d'éléments auront reçu une image.

Les contraintes du dernier type enfin subiront chacune, en leur temps, un traitement spécifique (II.3).

I.5 - LES FONCTIONS ECONOMIQUES.

Le premier type est celui où il n'y a tout simplement aucune fonction à optimiser : on cherche un objet ou encore tous les objets qui vérifient certaines propriétés combinatoires.

Le second type est courant en programmation mathématique où l'on cherche

(Z1) $\min Z$ avec

$$Z = P(x)$$

où P est un polynôme, éventuellement du premier degré (programmation linéaire), des inconnues x_i $(i=1,\ldots,D)$ et de coefficients connus.

Dans de nombreux problèmes de graphe c'est une quantité de la forme

(Z2) $$\min \sum_{i \in \mathcal{D}} c(i,f(i))$$

où $\mathcal{D}$ est l'ensemble des inconnues , f(i) la valeur par la fonction f cherchée de i et c une matrice de coûts donnée.

C'est le cas des problèmes de voyageur de commerce (minimum de la distance parcourue) , de flot maximum, de transports, de localisation.

Cette forme est également rencontrée sous deux variantes :

$$(Z3) \quad \min \max_{i \in \mathcal{D}} f(i)$$

$$(Z4) \quad \min \max_{i \in \mathcal{D}} (c(i) * f(i))$$

Elles apparaissent dans les problèmes de découpe (minimum de pièces utilisées), d'ordonnancement (durée au plus tôt d'achèvement des travaux), de coloration, de recouvrement.

Z1 min polynôme

Z2 $\min \sum_i c(i, f(i))$

Z3 $\min \max_i f(i)$

Z4 $\min \max_i c_i \cdot f(i)$

Z5 autres formes

Cinq cas d'optimisation

1.6 - ECHANTILLON D'ENONCES

Nous donnons ci-après 5 énoncés en ALICE.

Les problèmes correspondants sont de "petits" "toy-problems" qui n'ont qu'une valeur d'exemple. Des énoncés semblables mais de taille importante ont été également soumis au programme et seront décrits en partie 4 .

Ces cinq problèmes seront cependant "corrigés" en III.6.

PROBLEME 1 : Programmation mathématique en entiers.

Rappelons que l'on dénomme programme mathématique en entiers, tout problème mis sous la forme

$$\min c(x_1,x_2,\ldots,x_n)$$

sous les contraintes

$$a_i(x_1,x_2,\ldots,x_n) \geq 0 \quad \text{pour} \quad i=1,2,\ldots,m$$

où c et les a_i sont des fonctions à valeurs entières, linéaires ou non, de n variables entières $x_1,x_2,\ldots,x_n$.

Un exemple, tel qu'il se présente dans [Garfinkel et Nemhauser 1972] page 347, est

$$\min Z = 2x_1x_3 + 4x_2 + 3x_1x_5 + 2x_4 = 3x_5$$

avec

$$x_1x_4 + 3x_2x_5 - x_1 - 2x_3 - x_4x_5 - 1 \geq 0$$
$$2x_1 + 5x_3 - 2x_4 - 3x_5 - 2 \geq 0$$
$$x_1,x_2,\ldots,x_5 = 0,1$$

ce qui donne dans le langage ALICE, où les mots de l'énoncé formel sont sur trois caractères suivi d'un blanc :

```
CARF
****
SCI ENS VAR =   INT 1   5
        ECC =   INT 0   1
IRC FCN X   >-> VAR ECC
AVE MIN +   *   2   *   X   1   X   3   +   *   4   X   2   +   *   3   *   X
1   X   5   +   *   2   X   4   *   3   X   5
AVE <=  0   +   *   X   1   X   4   +   *   3   *   X   2   X   5   +   MNS X
1   +   MNS *   2   X   3   +   MNS *   X   4   X   5   MNS 1
AVE <=  0   +   *   2   X   1   +   *   5   X   3   +   *   MNS 2   X   .   +
MNS *   3   X   5   MNS 2
FIN
```

INT i j désigne l'ensemble des entiers successifs de i inclus à j inclus.

PROBLEME 2 : Puzzle Logique.

Il s'agit en fait d'un problème de couplage non trivial quoique d'allure mathématique amusante de type Lewis Carroll ; il était mentionné dans [Laurière 71] comme problème d'emploi du temps.

L'énoncé en langage naturel en est :

Cinq personnes de nationalités différentes habitent les cinq premières maisons d'une rue.

Elles exercent cinq professions différentes et ont chacune une boisson et un animal favori tous différents. Les cinq habitations sont de cinq couleurs différentes :

(1) - L'anglais habite la maison rouge.
(2) - L'espagnol possède un chien.
(3) - Le japonais est peintre.
(4) - L'italien boit du thé.
(5) - Le norvégien habite la première maison à gauche.
(6) - Le propriétaire de la maison verte boit du café.
(7) - Cette maison verte est à droite de la blanche.
(8) - Le sculpteur élève des escargots.
(9) - Le diplomate habite la maison jaune.
(10) - On boit du lait dans la maison du milieu.
(11) - Le norvégien habite à côté de la maison bleue.
(12) - Le violoniste boit des jus de fruits.
(13) - Le renard est dans la maison voisine du médecin.
(14) - Le cheval à côté de celle du diplomate.

Qui élève le zèbre et qui boit de l'eau ?

Numérotons les nationalités, professions, couleurs, boisson et animaux :

	1	2	3	4	5
nat.	anglais	espagnol	japonais	italien	norvégien
prof.	peintre	diplomate	violoniste	médecin	sculpteur
col.	vert	rouge	jaune	bleu	blanc
boi.	jus	eau	thé	café	lait
ani.	chien	zèbre	renard	escargot	cheval

```
EX2
***
SCI ENS N    =    INT 1    5
TRC BIJ NAT >-> N    N
        PRF >-> N    N
        CCL >-> N    N
        BCI >-> N    N
        ANI >-> N    N
AVE =   NAT 1    CCL 2
    =   NAT 2    ANI 1
    =   NAT 3    PRF 1
    =   NAT 4    BCI 3
    =   NAT 5    1
    =   CCL 1    BCI 4
    =   CCL 1    +    CCL 5    1
    =   PRF 5    ANI 4
    =   PRF 2    CCL 3
    =   3    BCI 5
    CL  =   NAT 5    -    CCL 4    1    =    NAT 5    +    CCL 4    1
    =   PRF 3    BCI 1
    CL  =   PRF 4    -    ANI 3    1    =    PRF 4    +    ANI 3    1
    CL  =   PRF 2    -    ANI 5    1    =    PRF 2    +    ANI 5    1
FIN
```

L'abréviation BIJ précise que l'on cherche des bijections de [1,5] dans [1,5] .

Il y a lieu de faire deux remarques à propos de cet énoncé :

1°) bien qu'il nous faille produire un petit effort pour poser mathématiquement le problème il n'y a besoin pour cela d'aucun artifice, il est simplement nécessaire de donner au programme l'énoncé dans sa totalité. Nous faisons en effet des inférences à la lecture du texte que le programme ne peut reconstruire "ex nihilo" : par exemple , rouge blanc et vert font partie de l'ensemble des couleurs; "à côté de" veut dire soit à droite soit à gauche.

2°) Le problème, s'il est maintenant rigoureusement posé, n'en est pas résolu pour autant.

PROBLEME 3 : Ordonnancement disjonctif.

Après avoir questionné les participants d'un congrès pour connaître les séances auxquelles ils souhaitent assister, on a dressé le tableau suivant pour les 11 séances :

	1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	0	1	0	1	1	0	1	0
2		1	1	1	0	0	1	1	1	0	1
3			1	0	1	1	0	1	0	0	0
4				1	0	1	0	1	0	1	0
5					1	0	1	0	1	0	1
6						1	1	0	0	1	0
7							1	1	1	0	0
8								1	1	0	1
9									1	1	0
10										1	0
11											1

dans lequel, l'élément i,j égal à 1 indique qu'un congressiste au moins souhaite suivre la séance i et la séance j (Cette matrice est évidemment symétrique).

De plus les organisateurs ne disposent que de trois salles dans lesquelles les sessions, qui sont toutes de même durée, peuvent se tenir en parallèle.

La séance 10 doit , enfin , impérativement avoir lieu après la 5 , la 11 après la 4 et la 6 .

On demande de répartir les séances en respectant ces contraintes et en minimisant la durée totale.

```
EX3
***
SCI CST N
    ENS S   =   INT 1   N                ensemble des séances.
        P   =   INT 1   N                ensemble des périodes.
TRC FCN F   >-> S   P   DIS DMA          disjonctions et salles.
AVE MIN MAX F   I                        durée minimale.
     <   F   5    F   10                                        } autres contraintes
     ET  >   F   11   F   4   >   F   11  F   6                 }
FIN
11,
2,3,5,7,8,10,0,0,0,0,0,0,0,0,0,0,0,0,0,
3,4,7,8,9,11,0,0,0,0,0,0,0,0,0,0,0,0,0,
5,6,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6,8,10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
7,9,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
7,10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
8,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3,3,3,3,3,3,3,3,3,3,3,
FIN
```

La première donnée numérique est la valeur de la constante N .

Les dix lignes suivantes sont associées au mot clé dis (pour disjonction) qui provoque leur lecture.

Cette abréviation est équivalente aux contraintes

$$\forall i, \forall j, j \in L_i \qquad f(i) \neq f(j)$$

où L_i est l'ensemble des éléments de la ligne i : deux sessions qui doivent être suivies par la même personne sont distribuées à des heures différentes.

Deux nouveaux mots du langage sont encore utilisés :

dma (pour degré maximum), associé à la dernière ligne des données est l'abréviation pour

$$\forall j \quad |\{i | f\ i = j\}| \leq 3 \quad \text{où} \quad |\ | \equiv \text{cardinal}$$

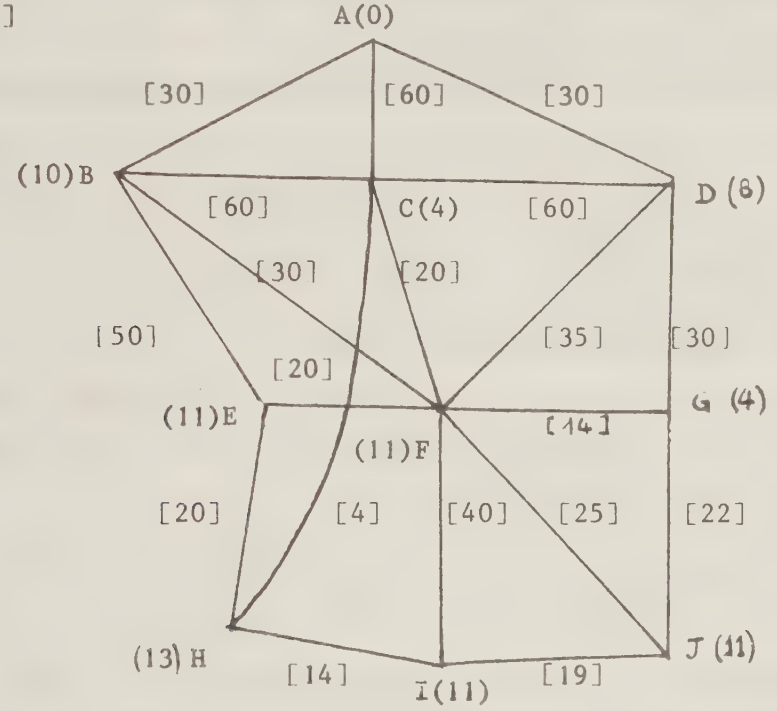
min max f i indique au programme que l'on cherche à minimiser l'ensemble image i.e.

$$\min_{i \in S} \max \ f(i)$$

PROBLEME 4 : Un problème de flot sur un arbre :

Soit $G = (X,U)$ le graphe représentatif du réseau des liaisons électriques à haute tension en France. On cherche en utilisant les arcs $u \in U$ de ce graphe, de capacités maximales données, à ravitailler en électricité les noeuds de distribution, sommets $x \in X$ du graphe. A tout x est associé une demande minimale, consommation en ce point, dem (x) , qu'il faut satisfaire.

Il s'agit donc de définir un flot [Berge 70] sur X qui doit, en outre, ne pas comporter de boucle : l'ensemble des arcs parcourus par un flot non nul forme un arbre. Les données numériques proviennent de la revue EDF [Huard et Ludot 1970]



- entre crochets, sur les arcs figure leur capacité maximale.
- entre parenthèses, sur les sommets leur consommation.

Il faut donc déterminer, pour tout sommet x , un sommet unique situé directement en amont, $f^{-1}(x)$. L'orientation de l'arc i j se fait alors de i vers j si f(i)=j. Le sommet A est en aval de tous les autres et la contrainte de capacité s'écrit :

$$\forall\, j \in X \quad \sum_{\substack{i \in X \\ \exists\, \text{Che } ij}} \text{dem}(i) \;\leq\; \text{val}\,(j, f(j))$$

($\exists$ Che ij = il existe un chemin de i à j , constitué d'arcs $\overrightarrow{if(i)}$ avec l'orientation qui vient d'être précisée).

```
    EX4
    ***
    SCI ENS SCM =    INT 1    1C
        CCE CEM SUR SCM
    TRC FCN F    >-> SCM SCM ARB SUC SYM VAL
    AVE <=  SIG ET  APP I    SCM CFE I    J    CEM I    VAL J    F    J
    FIN
    C,1C,4,8,11,11,4,13,11,11,
(1) 1,2,3,4,C,C,C,C,C,0,0,C,C,C,C,C,C,C,C,C,C,
(2) 3,5,6,C,C,C,0,C,C,0,0,C,C,C,C,C,C,C,C,C,C,
(3) 4,6,8,C,C,C,0,C,0,C,0,0,C,C,C,C,C,C,C,0,C,C,
    6,7,C,C,0,0,0,0,C,0,C,C,C,C,C,C,C,C,C,C,C,C,
    6,8,C,C,C,C,0,0,C,0,0,C,0,C,C,C,0,C,C,C,C,
    7,9,1C,C,0,0,0,0,C,C,0,0,0,0,C,C,C,0,0,C,C
    1C,C,C,C,0,C,C,C,C,C,0,0,C,C,C,0,0,C,C,0,C
    9,C,C,C,C,C,0,0,C,C,0,0,C,C,C,C,C,0,C,C,C,
    C,C,C,C,C,0,0,0,C,C,0,0,C,C,C,C,C,C,C,C,C,C,
    999,3C,60,30,6C,5C,50,60,2C,2C,35,30,4,2C,14,4C,25,22,14,19,
    FIN
```

Le graphe est ainsi donné par :

suc associé à l'ensemble des successeurs des sommets,

sym indique que ce graphe est symétrique. Ainsi les arcs issus d'un sommet sont lus ligne après ligne; les lignes (1) à (3) précisent donc que le sommet 3 est relié aux sommets 1 , 2 , 4 , 6 et 8 .

val est une option qui signifie ici que les arcs du graphe sont valués, les valuations étant données en dernière ligne (capacités des arcs).

arb (pour arbre) demande que le flot f forme un arbre, c'est-à-dire interdit la présence de circuits.

PROBLEME 5 - Un problème de recouvrement :

Considérons un ensemble $I = \{1,2,\ldots,n\}$ et une famille $P = \{P_1, P_2, \ldots, P_m\}$ de parties de I ,

$$P_j \subseteq I \quad \text{pour} \quad j \in J = \{1,2,\ldots,m\} .$$

Un sous ensemble J_o de J est un recouvrement de P si et seulement si

$$\bigcup_{j \in J_o} P_j = I$$

Etant donné un vecteur de poids C sur J , on cherchera par exemple en outre à minimiser

$$\sum_{j \in J_o} C(j)$$

Prenons l'exemple où m=5 , n=8 de [Garfinkel et Nemhauser 1970, p. 315]

$P_1 = \{1,2\}$, $P_2 = \{3,4,5\}$, $P_3 = \{5,6,7\}$, $P_4 = \{7,8\}$, $P_5 = \{2,4,6\}$

$C = (3,7,5,8,10,4,6,9)$

Il s'agit donc de trouver une fonction multivoque (FMU) de l'ensemble des m parties dans [1,n] . Chaque partie peut être envoyée sur chacun des éléments qu'elle contient (SUC) et si $j \in [1,n]$ est une image d'une partie, c'est aussi une image de toutes les parties qui le contiennent (contrainte ①).

La minimisation se fait sur les $j \in [1,n]$ qui appartiennent finalement à l'ensemble image (contrainte ②) .

Cette formulation en termes de fonction est moins lourde que celle utilisant n variables booléennes et m contraintes qui est habituellement donnée. Ce point sera discuté en IV.2.1.

```
REC
***
SCI CST N   M
    ENS EI  =   INT 1   M
        EJ  =   INT 1   N
    CCE C   SUR EJ
TRC FNL F   >-> EI  EJ  SUC
AVE ==> =   F   I   J   CQS APP L   PRE J   =   F   L   J                   ①
    MIN SIG ET  APP J   EJ  EXI APP I   EI  =   F   I   J   C   J       ②
FIN
8,5,
3,7,5,8,10,4,6,9,
1,2,C,C,0,C,0,C,0,C,C,0,C,C,C,C,C,0,C,C,
3,4,5,C,0,C,C,C,C,0,0,C,C,C,C,C,C,C,C,C,
5,6,7,C,C,C,0,C,0,0,0,C,C,C,C,C,C,C,C,C,
7,8,C,C,C,C,0,C,C,0,0,C,C,C,C,C,C,C,C,C,
2,4,6,C,C,0,0,C,C,0,C,C,0,C,C,C,C,0,C,C,
FIN
```

LES REPRESENTATIONS

L'information, relative à un énoncé donné, est essentiellement mémorisée et traitée sous deux formes :

- une représentation en graphe biparti.
- le calcul formel en polonaise.

II.1 - REPRESENTATION INTERNE SOUS FORME DE GRAPHE.

Lors de la lecture de l'énoncé formel d'un problème le programme traduit le maximum d'information dans une représentation interne.

Cette représentation est la plus naturelle qui soit, puisqu'il s'agit fondamentalement de la représentation sagittale d'une fonction ou, en d'autres termes, d'un graphe biparti :

A toute recherche de fonction, correspondant à l'énoncé

"trouver une fonction f E → F "

le programme associe un ensemble E de sommets de départ et un ensemble F de sommets images.

Une liste contient les images possibles pour tout i de E , une autre les antécédents pour tout j de F .

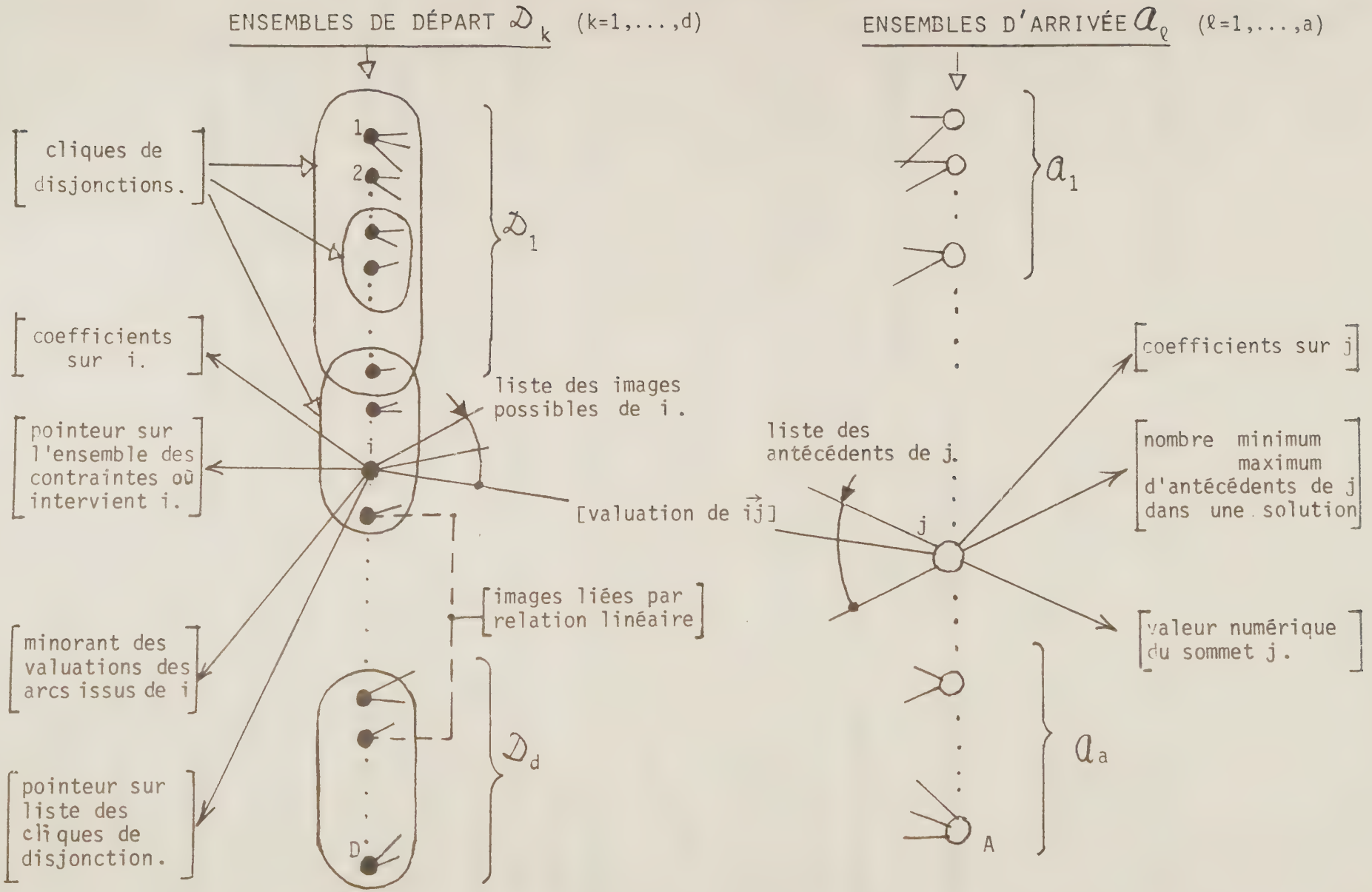
Si l'option "successeur" (SUC) est spécifiée lors de la déclaration de f , les successeurs des différents sommets de départ sont lus et seuls rangés dans la liste. Par défaut ce sont tous les j de F qui sont images possibles de tout i .

La page suivante décrit la façon dont sont traduites d'autres caractéristiques des fonctions à trouver, dans cette représentation sous forme de graphe.

En outre deux cas spéciaux concernant la nature de f sont prévus et mémorisés lorsque f n'est pas une fonction.

f peut être en effet :

- une fonction multivoque.



SCHEMA DU GRAPHE BIPARTI DE LA REPRESENTATION INTERNE DES PROBLEMES

(exemple : les problèmes de recouvrement dans la formulation de I .6.5)

- une restriction de E vers F , c'est-à-dire que seul un sous-ensemble E' de E , E' étant à déterminer, a une image par f .

(exemple : trouver un stable maximal dans un graphe)

Ces deux caractéristiques possibles de f , comme les autres informations relatives au graphe interne, vont être utilisées au moment des implications.

Dans le schéma les indications entre crochets sont facultatives :

a) Cliques de disjonction : deux sommets i_1 et i_2 dont on cherche à déterminer les images appartiennent à une clique de disjonction si et seulement si il existe entre leurs images j_1 et j_2 la contrainte $j_1 \neq j_2$. Celle-ci apparait par exemple chaque fois que l'on cherche une injection.

Plus généralement ces cliques permettent de mémoriser les disjonctions qui traduisent, par exemple, la non-ubiquité d'éléments, ou la coloration en deux couleurs distinctes de pays limitrophes.

b) Coefficients et valuations sur i, j, $\overrightarrow{ij}$: L'optimisation se fait souvent sur des coûts associés aux inconnues ou encore sur les coûts de l'affectation du sommet (de l'inconnue) i à l'image j . Ces coûts sont alors liés au graphe et utilisés par le programme pour l'optimisation.

c) Minorant des valuations des arcs issus de i : Ceci correspond à un cas particulier d'optimisation qui intervient fréquemment et sera notamment utilisé en III.4 .

d) Nombres minimum et maximum d'antécédents : De même que les cliques de disjonction généralisaient l'injection , ceci généralise la surjection ; dans le cas où l'on cherche à obtenir une surjection, ces deux nombres valent 1 en tout sommet arrivée. Si certains sommets ne doivent pas être nécessairement atteints leur nombre minimum d'antécédents est 0 .

Un nombre maximum d'antécédents égal à m pourra traduire par exemple le fait que tel moyen est disponible en au plus m exemplaires pendant une période de temps.

TABLEAU DE L'ENSEMBLE DES TYPES DE CONTRAINTES
DIRECTEMENT INTERPRETEES SOUS FORME DE GRAPHE INTERNE

$f(i) = \alpha$

$f(i) \lesssim \alpha$

$f(i) \neq f(j)$ (cas général : injection)

$|\{i/f(i) = j\}| \geq \alpha_j$ (surjection)

$|\{i/f(i) = j\}| \leq \beta_j$

$f(i) = \bigwedge g(k) + \nu$

$coef(i, f\ i) \lesssim \alpha$

II.2 IMPLICATIONS GEREES PAR LE GRAPHE INTERNE.

La représentation précédemment décrite sert essentiellement à déduire, à tout instant, le plus de renseignements possibles des "contraintes de structure" du problème.

Ainsi, en deux endroits, des implications sont détectées par le graphe: affectation d'un sommet, élimination d'un arc.

1 - Affectation: f(i) = j.

Lorsque le programme donne au sommet de départ i l'image j :

- il supprime (voir 2) d'abord tous les autres arcs issus de i s'il en existe et si f est une fonction (non multivoque).
- il supprime tous les autres arcs atteignant j s'il en existe et si le degré maximum de j est atteint.
  (par exemple f est une injection).
- il supprime les arcs atteignant j et venant d'un sommet i' qui est dans une clique de disjonction contenant i .
- il supprime enfin l'arc qui fermerait un circuit parasite dans le cas où l'on cherche un chemin élémentaire, un arbre ou un circuit hamiltonien.

Dans ces trois cas, le graphe, donné dans l'énoncé du problème par $G = (X,U)$ où X est l'ensemble de sommets, U l'ensemble des arcs, se retrouve ici sous forme de graphe biparti, X étant doublé :

$$E \equiv F \equiv X$$

et $i \in E$ est lié à $j \in F$ si et seulement si $\overrightarrow{ij} \in U$.

Exemple

Si l'on a déjà

$$f(2) = 4$$

lorsque l'on fait

$$f(1) = 2$$

il faut interdire le circuit parasite

1 - 2 - 4 - 1

en posant

$$f(4) \neq 1$$

Il est remarquable que ceci suffise à traiter globalement les différents cas : chemins, arbres et circuits hamiltoniens.

- Lorsque le programme traite un problème d'optimisation, l'affectation de i à j peut en outre donner naissance à un autre groupe d'implications .

- Si la somme des coûts (éventuellement pondérés) des affectations déjà faites et de

$$f(i) = j \quad \text{est supérieure à}$$

une borne supérieure connue, il est inutile de continuer dans cette voie : on retourne à la gestion de l'arborescence des choix (cf.III) pour redémarrer différemment.

Remarque :

Certaines contraintes algébriques, contenant un signe somme , telles que :

$$\sum_{\substack{i \in \mathcal{D} \\ fi=j}} \text{coef}\ (i) \leq L_j$$

(la somme des coefficients attachés à tous les sommets i ayant la même image doit être inférieure à une constante L_j connue), reviennent souvent en recherche opérationnelle, et sont commodément gérées également lors d'une affectation dans le graphe interne.

Ceci sera développé en II.3.

2 - Suppression d'un arc $\vec{ij}$:

Lorsque le programme supprime l'arc reliant le sommet i au sommet j

- il mémorise cette suppression dans une pile dont le pointeur est, en général associé au dernier choix qui a conduit à cette suppression.

- si ij était le dernier arc issu de i (resp. atteignant j) et que l'on cherche une fonction (resp. une surjection), le contexte dans lequel on se

trouve ne peut contenir de solution : il y a régression.

- si il n'y a plus qu'un arc partant de i après cette suppression, on a une affectation forcée.

- si le degré minimum est atteint pour le sommet j, il faut procéder de même à un ensemble d'affectations sans choix.

- enfin, si tous les arcs sont valués et si l'on cherche à minimiser une fonctionnelle dépendant de ces valuations, il y a lieu de recalculer le coût minimum de l'affectation du sommet i à une image quelconque autre que j.

II.3. MANIPULATION FORMELLE DES EXPRESSIONS.

Les contraintes algébriques classiques telles que :

(i) $x_1 \cdot x_2 - 6x_3 + x_4 \geq 0$ ou bien

(ii) $x_1 = 3 \Rightarrow x_2 = 1 - x_3$ ou encore

(iii) $\sum_{i \in I} \text{coût}\ (i, f(i)) \leq 52$

ne sauraient être directement interprétées dans le graphe interne.

Elles sont donc conservées et manipulées telles quelles.

Plus précisément, pour avoir une notation linéaire, et pour des raisons de technique de programmation, (ceci sera discuté en IV.2) elles sont traitées par le programme dans la forme polonaise préfixée ((i) devenant $\geq + * x_1 x_2 + \text{mns} * 6x_3 x_4\ 0$) mais nous garderons ici par commodité de lecture, la notation habituelle.

Il s'agit alors pour le programme de déduire, à chaque instant, de ces contraintes algébriques l'information maximale, comme il le fait dans la représentation graphique.

Pour ce faire, il étudie les contraintes une à une, les normalise, résout certaines d'entre elles localement (équations diophantiennes simples), engendre de nouvelles contraintes nécessaires pour satisfaire telle contrainte primaire, combine enfin les contraintes entre elles.

II.3.1. Standardisation des expressions.

L'algorithme d'unification [PITRAT 1966] permet d'appliquer à toute expression toute règle de réécriture par substitutions successives.

Ainsi la règle $t + t \rightarrow 2\ t$ appliquée à

$3.x_4 + 3.x_4$ "simplifiera"

cette expression en $2.3.x_4$.

Toutes les règles de réécriture utilisées, dont la liste exhaustive est en annexe 2, sont obligatoires, en ce sens que dès que l'algorithme d'unification détecte l'application possible de l'une d'entre elles, il modifie définitivement l'ancienne expression et l'oublie : toutes ces règles sont des règles de normalisation.

Standardisation des contraintes en $=$, $\leq$, $\geq$, $>$, $<$, $\neq$.

Le programme fait systématiquement passer tous les termes non nuls dans le membre de droite. En outre ces contraintes sont toujours ramenées aux seules connectives principales $=$, $\leq$ et $\neq$. Ainsi

$x > y$ est réécrit $0 \leq x - y - 1$

Traitements de $+$, $*$, $-$:

Seul le "moins unaire" est conservé par le programme

$x - y$ devenant $x + MNS\ y$,

la distributivité de la multiplication est appliquée partout et l'associativité du $+$ et du $*$ donne lieu à la forme normalisée :

$$t_1 \perp t_2 \perp \ldots \perp t_n$$

où les t_i sont des termes quelconques, $\perp$ désignant $+$ ou $*$.

A l'intérieur d'une telle expression ne contenant donc que des $+$ ou que des $*$, les termes sont eux mêmes réordonnés suivant un "hash code" par poids monotones,

$0 = x_1 + -6x_1 . x_3 + 4 + x_7 + x_2 . (1 + -x_4 . x_7) - x_2$

devenant

$$0 = 4 + x_1 + x_2 + \text{mns} x_2 + x_7 + \text{mns}\ 6x_1 \cdot x_3 + \text{mns}\ x_2 \cdot x_4 \cdot x_7$$

toutes les simplifications habituelles (calcul sur les éléments neutres et les constantes) sont alors évidemment appliquées.

Symboles OU , ET , ¬ :

Toute expression dont la connective principale est un ET est séparée en deux expressions.

L'associativité du OU est traitée comme celle du + ou du * , les sous-expressions commandées par des OU étant elles-mêmes normalisées.

De plus, chaque sous-expression, lors de la standardisation, d'une contrainte OU est examinée pour voir si, dans le contexte, elle n'est pas trivialement satisfaite ou,au contraire, contredite; dans l'un ou l'autre cas le programme détruit simplement la contrainte p ou vrai ,réécrit p ou faux en p.

Les négations des propositions sont obtenues et simplifiées par unification (cf. page 9 par exemple).

Symboles Σ et $\forall$:

Les expressions $\sum_{i \in E} t(i)$, $\forall\, i \in E\ \mathcal{R}(i)$ sont, en général, développées dès que E est connu : pour le "quel que soit" on obtient autant de nouvelles contraintes qu'il y a d'éléments dans E .

Il existe cependant plusieurs cas spéciaux où des contraintes sont étudiées par le programme avant que E ne soit totalement connu; ces cas ne relèvent plus à proprement parler de la normalisation et seront developpés plus spécialement en II.3.

Les deux pages suivantes donnent la liste des règles de réécriture utilisées par l'algorithme d'unification.

Le symbole --> doit se lire "se réécrit" et les règles apparaissent sous la forme :

--> membre gauche membre droit

Les symboles OSi (i=1 à 4) correspondent à des groupements d'opérateurs; ils s'unifient si l'un des opérateurs du groupe coïncide avec une connective de l'expression .

OS1	+	*	/	MOD	OS2	=	¬=	
OS3	=	¬=	<=		OS4	=	-->	

Ceci évite une certaine redondance dans la liste des règles.

Les symboles CSP i (i=1 à 22) renvoient à autant de branchements particuliers à l'intérieur de l'algorithme.

De façon à réduire les tentatives d'unification,les règles ne sont appelées que par le jeu de déclencheurs.

Pour chaque règle les caractéristiques suivantes du membre gauche sont mémorisées :

- nom de la connective principale
- nom de la connective éventuelle suivant immédiatement la connective principale
- autres connectives
- présence de 0
- présence de 1
- présence d'une constante numérique (CNU ou CNV dans les tables)
- présence d'un signe de fonction (FFF ou GGG dans les tables)
- double occurrence d'un même terme par exemple --> + x mns x 0

Lorsque le programme travaille sur une expression,un profil relatif aux sept points précédents est calculé et seules les règles de même profil sont essayées. En outre quand l'unification avec une règle donnée est réussie, l'essai suivant est indiqué par un pointeur déterminé empiriquement et associé à la règle.

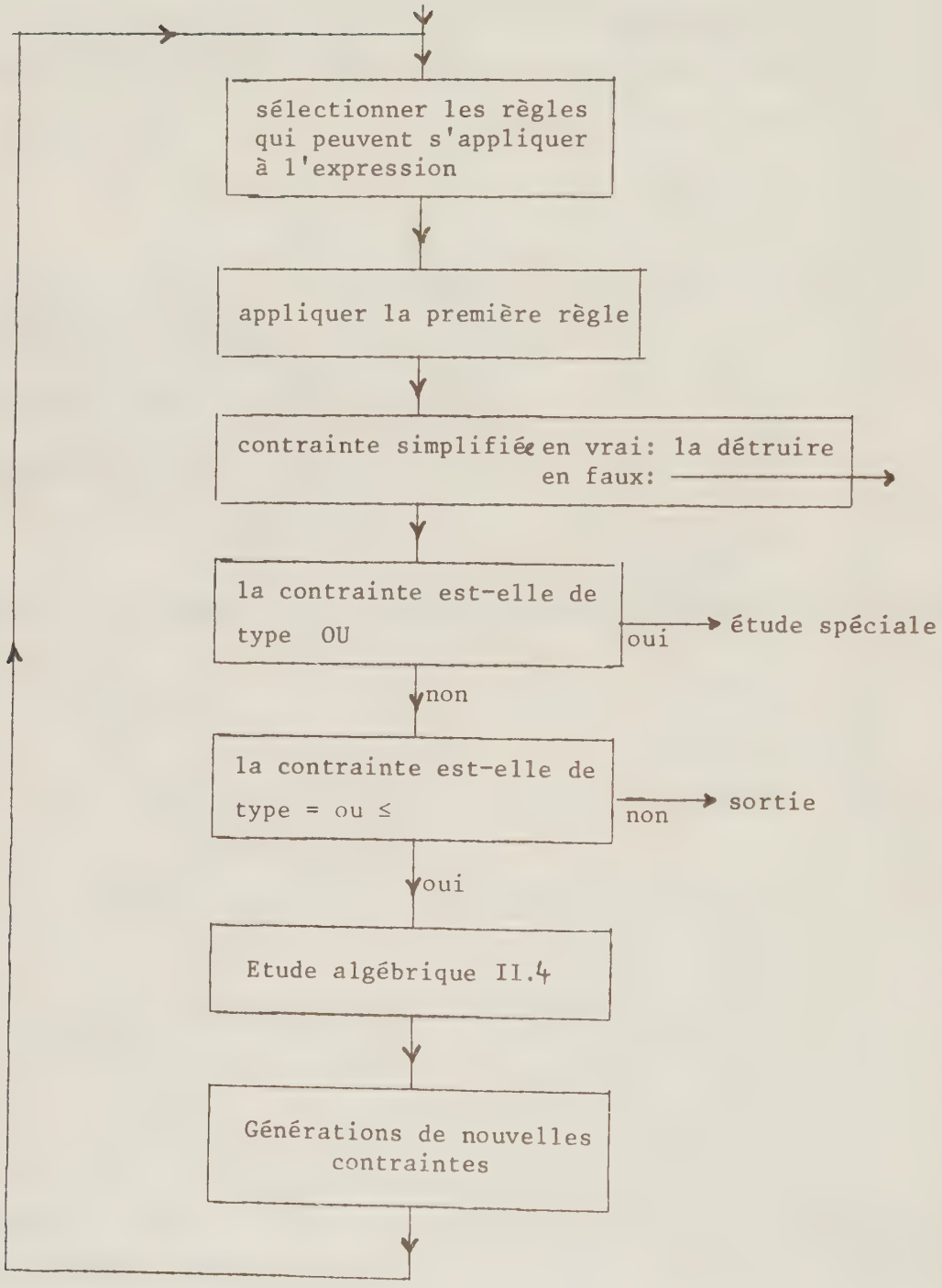
```
--> CS5 XNC YYY CS5 C   +   MNS XNO YYY
--> >   XXX YYY <=  C   +   MNS 1   +   XXX MNS YYY
--> <   XXX YYY <=  C   +   MNS 1   +   MNS XXX YYY
--> >=  XXX YYY <=  C   +   XXX MNS YYY
--> -   XXX YYY +   XXX MNS YYY
--> CS2 C   MNS XXX CS2 C   XXX
--> +   +   XXX YYY ZZZ +   XXX +   YYY ZZZ
--> SIG APP IND EEE XXX CSP 13  SIG APP IND EEE XXX
--> *   *   XXX YYY ZZZ *   XXX *   YYY ZZZ
--> +   XXX MNS XXX C
--> +   XXX +   MNS XXX YYY YYY
--> <=  C   +   CNU *   CNV XXX CSP 14  <=  CNL *   CNV XXX
--> <=  C   +   CNU *   XXX YYY CSP 18  +   CNL *   XXX YYY
--> *   XXX /   YYY XXX YYY
--> *   MNS XXX /   YYY XXX MNS YYY
--> MNS /   XXX ZZZ /   MNS XXX ZZZ
--> /   XXX MNS ZZZ /   MNS XXX ZZZ
--> /   *   XXX YYY ZZZ *   /   XXX ZZZ YYY
--> +   /   XXX ZZZ /   YYY ZZZ /   +   XXX YYY ZZZ
--> +   XXX XXX *   2   XXX
--> +   XXX +   XXX YYY +   *   2   XXX YYY
--> +   XXX *   CNV XXX *   +   1   CNV XXX
--> +   MNS XXX *   CNV XXX *   +   MNS 1   CNV XXX
--> +   XXX +   *   CNV XXX YYY +   *   +   1   CNV XXX YYY
--> +   MNS XXX +   *   CNV XXX YYY +   *   +   MNS 1   CNV XXX YYY
--> +   *   CNU XXX *   CNV XXX *   +   CNU CNV XXX
--> +   *   CNL XXX +   *   CNV XXX YYY +   *   +   CNL CNV XXX YYY
--> MNS +   XXX YYY +   MNS XXX MNS YYY
--> <=  C   MNS XXX =   C   XXX
--> *   MNS CNU +   XXX YYY +   *   MNS CNU XXX *   MNS CNL YYY
--> *   CNU +   XXX YYY +   *   CNU XXX *   CNL YYY
--> MNS *   XXX YYY *   MNS XXX YYY
--> *   XXX MNS YYY *   MNS XXX YYY
--> *   XXX *   MNS YYY ZZZ *   MNS XXX *   YYY ZZZ
--> CS1 CNU CS1 CNV XXX CS1 CS1 CNU CNV XXX
--> CS1 CNL CNV CSP 1   CS1 CNU CNV
--> *   1   XXX XXX
--> *   MNS 1   XXX MNS XXX
--> MNS C   C
--> *   C   XXX C
--> +   C   XXX XXX
--> MNS MNS XXX XXX
```

```
--> CS2 C   +   CNU MNS YYY CS2 C   +   MNS CNL YYY
--> CS2 C   +   CNU +   MNS YYY ZZZ CS2 C   +   MNS CNL +   YYY MNS ZZZ
--> CS2 C   +   CNU *   MNS CNV YYY CS2 C   +   MNS CNL *   CNV YYY
--> CS3 C   FFF CNV CSP 6   CS3 C   FFF CNV
--> CS3 C   +   CNU FFF CNV CSP 7   CS3 CNL FFF CNV
--> CS3 C   CNU CSP 2   CS3 CNU C
--> =   C   *   CNU FFF CNV =   C   FFF CNV
--> --> FFF CNV MNS FFF YYY =   C   FFF YYY
--> QQS APP INC EEE XXX CSP 11  QQS APP INC EEE XXX
--> QQS APP INC INT CNL CNV XXX CSP 21  QQS APP INC INT CNL CNV XXX
--> QQS APP INC PRE CNL XXX CSP 17  QQS APP INC PRE CNL XXX
--> QQS APP INC SLC CNL XXX CSP 17  QQS APP INC SLC CNL XXX
--> COE CNL CSP 8   COE CNL
--> COB CNL CNV CSP 9   COB CNU CNV
--> <=  C   +   CNU MNS FFF CNV CSP 7   <=  CNL MNS FFF CNV
--> =   C   +   MNS FFF CNV CQQ YYY CSP 4   =   FFF CNV QQQ YYY
--> =   C   +   FFF CNV MNS QQQ YYY CSP 4   =   FFF CNV QQQ YYY
--> CS3 C   +   CNU *   CNV FFF XXX CSP 3   CS3 CNL *   CNV FFF XXX
--> CS2 C   +   CNU COE XXX CSP 15  CS2 CNL COE XXX
--> CS2 C   +   CNU COB CNV XXX CSP 16  CS2 CNL COB CNV XXX
--> ==> CLI XXX XXX
--> ==> NCN XXX CLI
--> <=> CLI XXX XXX
--> <=> NCN XXX ¬   XXX
--> ¬   QQS XXX YYY EXI XXX ¬   YYY
--> -   EXI XXX YYY QQS XXX ¬   YYY
--> ¬   CL  XXX YYY ET  ¬   XXX ¬   YYY
--> ¬   ET  XXX YYY CL  XXX YYY
--> ¬   =   XXX YYY ¬=  XXX YYY
--> ¬   <=  C   XXX <=  C   +   MNS 1   MNS XXX
--> CL  CLI XXX CLI
--> CL  XXX CLI CLI
--> CL  NCN XXX XXX
--> CL  XXX NCN XXX
--> CS4 CNL CNL CLI
--> CS4 CNL CNV NCN
--> --> CNL FFF CNV =   C   +   MNS CNL FFF CNV
--> --> CNL +   CNV FFF XXX =   C   +   MNS CNL +   CNV FFF XXX
--> --> XXX XXX CLI
--> CS3 FFF CNV C   CSP 6   CS3 C   FFF CNV
--> CS3 FFF CNV CNU CSP 7   CS3 MNS CNL FFF CNV
--> --> FFF CNV +   CNU FFF YYY CSP 22  --> FFF CNV +   CNL FFF YYY
```

	Expressions	Expressions normalisées
= <= <	$-f(1) + f(2) = 5 + f(3)$	$0 = 5 + f(1) + -f(2) + f(3)$
	$f(1) + g(2) < 6$	$0 \leq 5 + -f(1) + -g(2)$
	$4 \leq 3$	faux
	$g(1) = 3$	$g(1) \rightarrow 3$: toutes les occurences de g(1) sont remplacées par 3
	$0 \leq \alpha x + -\beta$	$0 \leq -\left\lceil \frac{\beta}{\alpha} \right\rceil + x$ (partie entière supér.)
	$0 \leq -\alpha x + \beta$	$0 \leq +\left\lfloor \frac{\beta}{\alpha} \right\rfloor - x$ (partie entière infér.)
	$3*f(4) < 6$	$0 \leq 1 + -f(4)$
¬ ou ∀ Σ	$\neg\neg\, g(1) = 3$	$g(1) \rightarrow 3$
	$\neg\, f(2) < 6$	$0 \leq -6 + f(2)$
	$\neg(f(2)=g(1)$ et $g(2)=1)$	$(0 \neg= f(2) + -g(1))$ ou $(0 \neg= 1 + -g(2))$
	$\forall\, i \in [1,2]\ f(i) = g(i)$	$f(1) \rightarrow g(1)$ et $f(2) \rightarrow g(2)$

Les contraintes qui ne comportent qu'une inconnue sont donc ainsi éliminées ; les unes par réécriture : $g(1) \rightarrow 3$, les autres $0 \leq -5 + f(2)$ par l'intermédiaire du graphe interne , qui supprime ici toutes les images possibles par f , du sommet 2 qui ont une valeur numérique inférieure à 5 .

Décrivons maintenant le sous-programme purement algorithmique, qui ayant pour entrée une contrainte algébrique, donne, en sortie, de nouvelles contraintes qui sont nécessairement satisfaites quand la première l'est.



SIMPLIFICATION ET ANALYSE DES CONTRAINTES

II.4 - TRAITEMENT ALGORITHMIQUE DES CONTRAINTES ALGEBRIQUES.

La contrainte, soit $\mathscr{C}$, se présente sous la forme normalisée :

$$0 \ \copyright \ \sum_{i \in T} t_i$$

où © désigne soit = soit ≤ .

Le principe est de comparer l'intervalle de variation de la somme des termes positifs d'une part, avec l'intervalle de variation de la somme des termes négatifs d'autre part. (Ces intervalles sont obtenus par simple sommation des valeurs minimales et maximales de chacun des termes, elles-mêmes données par le graphe). Soient [P,PP] et [N,NN] ces deux intervalles.

Exemple : Si la contrainte $\mathscr{C}$ est

$$0 = -9 - f(6) + 2*f(1) + g(3) \qquad \text{avec}$$
$$2 \leq f(1) \leq 5 \ , \ 0 \leq f(6) \leq 9 \ , \ 0 \leq g(3) \leq 2$$

il vient

$$[N,NN] = [9,18] \quad \text{et} \quad [P,PP] = [4,12]$$

Suivant les positions relatives de N,NN , P et PP et la connective principale (≤ ou bien =) , il y a 8 cas à distinguer.

Posons A = max (P,N) et B = min (PP,NN).

1. A > B

Si © est ≤ et si A = P alors $\mathscr{C}$ est trivialement satisfaite.#

(Le symbole # signifie la fin de traitement de $\mathscr{C}$)

Sinon $\mathscr{C}$ ne peut être satisfaite #

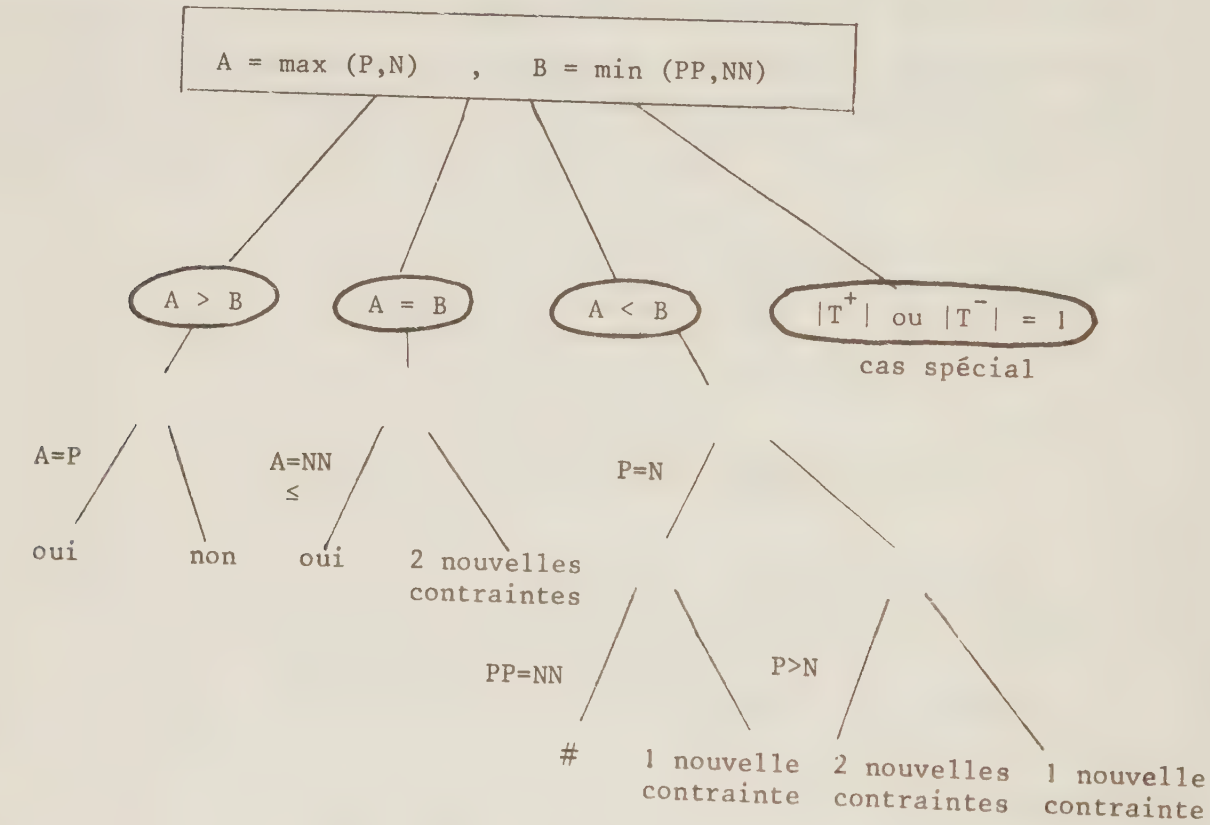
2. A = B

Si © est ≤ et si A = P = NN , $\mathscr{C}$ est trivialement satisfaite #

Sinon deux nouvelles contraintes sont engendrées :

$$\sum_{i \in T^+} t_i = A$$

T^+ = {termes positifs} T^- = {termes négatifs}	$P = \min \sum_{i \in T^+} t_i$ $N = \min(- \sum_{i \in T^-} t_i)$	$PP = \max \sum_{i \in T^+} t_i$ $NN = \max(- \sum_{i \in T^-} t_i)$



SCHEMA RESUMANT L'ANALYSE DE $\sum_{i \in T^+} t_i + \sum_{i \in T^-} t_i \geqq 0$

$$- \sum_{i \in T^-} t_i = A$$

où T^+ (resp. T^-) désigne le sous-ensemble de T des termes positifs (resp. négatifs).

3. $A < B$

(3.1) $P=N$

Si $PP = NN$, la contrainte est conservée telle quelle # .
Sinon , on engendre la contrainte

$$\sum_{i \in T^+} t_i \leq B \quad , \quad \text{si} \quad B = NN$$

$$- \sum_{i \in T^-} t_i \leq B \quad , \quad \text{si} \quad B = PP \#$$

(3.2) $P > N$ et $\mathcal{C}$ est en $\leq$ alors faire comme en 3.1.

(3.3) Sinon,on a la nouvelle contrainte

$$\sum_{i \in T^+} t_i \geq A \quad , \quad \text{si} \quad A = N$$

$$- \sum_{i \in T^-} t_i \geq A \quad , \quad \text{si} \quad A = P$$

Faire ensuite comme en 3.1.

4. Cas particulier.

Un cas particulier important est celui où

$$|T^+| \leq 1 \quad \text{ou bien} \quad |T^-| \leq 1$$

Les deux cas se traitent de façon similaire.

Prenons pour fixer les idées $|T^+| \leq 1$.

Le cas où $|T^+| = 0$ donne une contrainte trivialement soit vraie soit fausse # .

$|T^+| = 1$ conduit à fabriquer autant de nouvelles contraintes qu'il y a d'éléments dans T^- si © est $\leq$, 2 fois ce nombre sinon.

Soit en effet t_{i_o} un terme de T^- , on écrit

$$- t_{i_o^-} \leq PP - N_{i_o^-}$$

où PP est la valeur maximale du seul terme positif et $N_{i_o^-}$ la somme des valeurs minimales des termes de T^- différents de $t_{i_o^-}$.

Si $\mathscr{C}$ est une égalité, il vient en outre de la même façon :

$$- t_{i_o^-} \leq - P + NN_{i_o^-}$$

P = valeur minimale du terme positif,

$NN_{i_o^-} = \Sigma$ valeur maximale des $t_i \in T$, $i \neq i_o^-$.

Plusieurs exemples d'utilisation de cet algorithme sont donnés dans la partie IV .

L'ordre dans lequel les contraintes $\mathscr{C}$ sont examinées est important, ceci est du ressort du programme principal et est défini en II.3.

Remarquons simplement, qu'au contraire des méthodes habituelles en programmation linéaire en entiers, le nombre des contraintes varie ici continuellement et ceci joue un grand rôle et donne une efficacité certaine comme on le verra dans les résultats IV. En effet, au lieu de se contenter de se livrer à des tests ou encore à des calculs sur des pénalités associées aux variables, on adjoint effectivement les nouvelles contraintes engendrées, qui seront elles-mêmes, en leur temps, traitées de la même manière.

Cas spéciaux pour la contrainte étudiée $\mathscr{C}$:

A) Equations diophantiennes :

$\mathscr{C}$ est de la forme :

$$0 = \alpha + \beta f(1) + \gamma g(2)$$

une procédure teste toutes les valeurs possible de $f(1)$ et ne retient que les valeurs de $g(2)$ compatibles avec $\mathscr{C}$ puis inversement . Dans le cas où l'un des quotients $\frac{\alpha}{\beta}$, $\frac{\alpha}{\gamma}$, $\frac{\beta}{\gamma}$ ou $\frac{\gamma}{\beta}$ est entier une nouvelle règle de réécriture est de plus engendrée, qui exprime une inconnue en fonction de l'autre.

B) Elimination définitive d'inconnues.

Quand on a affaire à un système linéaire "petit" (au sens de III.3), et qu'en outre chaque inconnue apparait dans au moins deux contraintes distinctes, le programme élimine les inconnues les moins contraintes les unes après les autres, en les exprimant en fonction des plus contraintes.

C) Cas des retenues .

$\mathscr{C}$ est de la forme :

$$0 = t - x(1) + 10*y(1)$$

où t est une somme de termes dans laquelle chaque inconnue apparaissant doit être comprise entre 0 et 9 , ainsi que x(1) et y(1). Ceci s'entend évidemment en système décimal. $\mathscr{C}$ est alors réécrite en

$$y(1) \rightarrow \left\lfloor \frac{x(1) - t}{10} \right\rfloor$$

$$x(1) \rightarrow t \text{ modulo } 10 .$$

Dans chacun de ces trois cas , à chaque fois qu'une nouvelle règle de type $x \rightarrow \alpha$ est créée, deux nouvelles contraintes lui sont associées :

$$\text{minimum } x \leq \alpha \leq \text{maximum } x .$$

Les deux dernières méthodes ne sont pas toujours bonnes à utiliser. En effet, si elles réduisent le nombre d'inconnues, elles donnent concurremment naissance à des contraintes de plus en plus complexes et conduisent finalement à une perte d'information ou encore à des coefficients de grandeurs rédhibitoires [Bradley 1971] . Aussi le programme ne les emploie-t-il que lorsque la taille du problème, par des implications ou/et des choix, est descendue en dessous d'un certain seuil (cf. III.4).

Après lecture de l'énoncé, toutes les contraintes sont traitées une fois algorithmiquement comme il vient d'être décrit en II.1, II.2 et II.3.

Ce traitement a lieu en rangeant les contraintes par ordre de "simplicité" .

II.5 - ORDRE DE SIMPLICITE DES CONTRAINTES :

Donnant un poids unité à toute variable
un poids nul à la connective =
un poids 3 à la connective ≤
un poids 7 à toute autre connective

le programme analyse les contraintes dans l'ordre des sommes décroissantes des poids (complexités). En cas d'égalité c'est la contrainte dont le terme constant ("second membre") est le plus grand, qui est d'abord retenue.

Fonction de Grundy associée aux contraintes d'inégalités :

Soit $\{\mathscr{C}_k, k = 1,2,\ldots,K\}$ l'ensemble des contraintes d'inégalité pour un problème donné à un instant donné.

Chacune de ces contraintes est normalisée sous la forme

$$\sum_{i \in T^+_{\mathscr{C}_k}} t_i - \sum_{j \in T^-_{\mathscr{C}_k}} t_j \geq 0$$

Nous allons faire correspondre à toute inconnue x , $x \in \mathcal{D}$, une fonction $\mathscr{G}(x)$ à valeurs entières par l'algorithme suivant.

$$\forall\, x \in \mathcal{D} \ , \quad \mathscr{G}(x) := 1$$

(A) : faire pour $k = 1,2,\ldots,K$

$$\forall\, x, x \text{ apparaissant dans un } t_i \text{ avec } i \in T^+_k \ ,$$

$$\mathscr{G}(x) := \mathscr{G}(x) + \max_{y \text{ dans } T^-_k} \mathscr{G}(y)$$

Aller en (A) tant que

$$\forall\, x \ \mathscr{G}(x) \leq K \quad \text{et}$$
$$\exists\, x \ \mathscr{G}(x) \text{ a été modifié depuis la précédente itération.}$$

$\mathscr{G}(x)$ peut être regardé comme la longueur du chemin de longueur maximale reliant l'origine à x dans le graphe orienté $(\mathcal{D},U)$ associé aux contraintes d'inégalités avec

$$(x,y) \in U \Leftrightarrow \exists\, k \text{ , } x \text{ est dans un } t_i \text{ , } i \in T^-_k$$
$$y \text{ est dans un } t_j \text{ , } j \in T^+_k$$

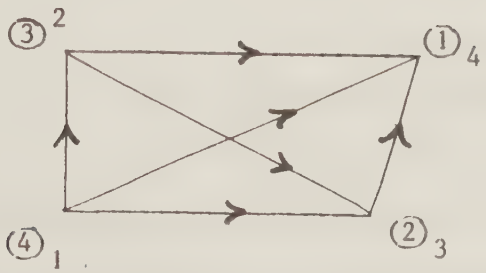
Exemple : K = 3

$\mathcal{C}_1$ $\quad f(1) + f(2) - [f(3) + f(4) + 3] \geq 0$

$\mathcal{C}_2$ $\quad f(1) - [f(2) + 5] \geq 0$

$\mathcal{C}_3$ $\quad f(3) - [f(4) + 2] \geq 0$

Le graphe $(\mathcal{X},U)$ est alors :



$\textcircled{x}_g$: sommet x de fonction $\mathcal{G}(x) = g$.

x	1	2	3	4
$\mathcal{G}(x)$	4	3	2	1

Il est alors clair que si

$$\exists\, x \quad \mathcal{G}(x) > K$$

il existe dans $(\mathcal{X},U)$ un circuit passant par le sommet x et le système des $\mathcal{C}_k$ n'a pas de solution finie.

Sinon on étudiera le système en s'intéressant aux contraintes une à une, en prenant à chaque fois la contrainte non encore étudiée pour laquelle on a :

(1) $$\min_k \; \max_{x \text{ dans } T_k} \mathcal{G}(x)$$

Ainsi, dans l'exemple, l'ordre $\mathcal{C}_3$, $\mathcal{C}_1$ et enfin $\mathcal{C}_2$ conduira successivement à

$f(3) \geq 2 + \alpha$ avec $\alpha = \min f(4)$

$f(2) \geq 5 + 2\alpha$

$f(1) \geq 10 + 2\alpha$

Quand il y a égalité pour plusieurs contraintes vis à vis du critère (1), la préférence sera donnée à la contrainte de complexité minimale.

II.6. TRAITEMENT DES CONTRAINTES DU TROISIEME TYPE ET TROISIEME REPRESENTATION

A toute contrainte de type :

$$\mathcal{Q} \Rightarrow \mathcal{P} \quad \text{où } \mathcal{Q} \text{ et } \mathcal{P} \text{ sont des expressions logiques,}$$

il est naturel d'associer le test :

$\mathcal{P}$ est-elle vraie ?

et d'écrire : $\neg\mathcal{P} \Rightarrow \neg\mathcal{Q}$ (II.6.1)

Le type de contrainte présenté en I.4.b , entre dans ce cadre. En effet, les objets i , qui ont en commun une certaine propriété $\mathcal{R}$, devaient satisfaire une certaine relation; on est donc dans le cas où l'existence d'un objet i_o, pour lequel cette relation n'est pas satisfaite, entraîne :

i_o ne peut pas posséder la propriété $\mathcal{R}$.

Exemple : Ainsi, si après avoir fixé la même image j_o à un sous ensemble I de points, il existe un $i_o \notin I$ tel que

$$\sum_{i \in I} \text{coef}(i) + \text{coef}(i_o) > \alpha$$

la contrainte

$$\sum_{\{i \mid fi=j\}} \text{coef}(i) \leq \alpha$$

entraine

$$\underline{\underline{f(i_o) \neq j_o}}$$

Ce troisième type de contraintes peut donc être pris en compte une fois pour toutes sous forme d'un test.

La forme générale :

$$\sum_{\{i \mid \mathcal{R}(i)\}} \text{coef}(i) \leq \alpha_{\mathcal{R}}$$

où $\mathcal{R}(i)$ est une certaine propriété et $\alpha_{\mathcal{R}}$ une constante connue, est souvent rencontrée (I.4).

Les cas, correspondants à des variantes simples et jugées suffisamment courantes, sont prévus,

par exemple $\mathcal{R}(i) \equiv f(i) =$ constante connue

$\mathcal{R}(i) \equiv \exists$ chemin de i à un sommet connu.

Si une contrainte de ce type figure dans un énoncé, elle fixe donc simplement la valeur d'un marqueur et de quelques paramètres éventuels (notamment $\alpha_{\mathcal{R}}$) . La satisfaction de la contrainte est directement gérée par des instructions correspondantes dans le programme. La contrainte est donc "représentée" par une instruction de test de type (II.6.1) ; cette instruction est exécutée après toute affectation.

Quand dans un énoncé apparait une contrainte de type $\mathcal{Q} \Rightarrow \mathcal{S}$ non prévue, cette contrainte reste sous sa forme primitive. Elle est simplement interprétée par unification, lors de chaque affectation. Il s'ensuit un ralentissement dans l'éxécution du programme. Une méthode pour pallier cet inconvénient est décrite en V.2 .

II.7. LES LIAISONS ENTRE LE GRAPHE ET LA MANIPULATION FORMELLE.

Un problème de recherche opérationnelle est,dans la programmation habituelle, traduit par une seule représentation (cela a été souligné en I.5) : le "modèle" peut être un programme en nombres entiers, un graphe, un hypergraphe, un matroïde ou plus généralement un programme dans lequel toutes les caractéristiques du problème, et ses contraintes figurent, les unes sous forme de tableaux, les autres sous forme d'instructions.

L'art du chercheur opérationnel est dès lors de trouver la représentation qui traduit les contraintes les plus importantes, quitte à ajouter, plus ou moins aisément à cette formulation de base les contraintes qui restent.

Ainsi le célèbre problème du voyageur de commerce se laisse mettre,pour

une part, aisément sous forme linéaire :

$$\forall j \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall i \quad \sum_{j=1}^{n} x_{ij} = 1$$

avec $X = [1,\ldots,n]$ ensemble des sommets

$x_{ij} = 0$ ou 1

Traduire alors les contraintes de cheminement, de façon à éliminer les circuits de longueur inférieure à n conduit à écrire (2^n-2) inéquations supplémentaires comme

$$\sum_{i \in Q} \sum_{j \in X-Q} x_{ij} \geq 1 \qquad \forall\, Q \subsetneq X,\ Q \neq \emptyset$$

Et, bien que ce modèle figure dans tout livre de recherche opérationnelle, ce n'est guère précisément une formulation opérationnelle. On préférera donc traduire ces contraintes par des tests (de boucles et de fourches [Berthier 1967]).

Mais si maintenant on impose au voyageur des contraintes qui relèvent de la formulation algébrique, il faudra les programmer une à une ce qui retire toute généralité au modèle. On ne sait d'ailleurs guère traiter ces problèmes qui font apparaitre des contraintes de natures différentes qu'il s'agisse d'ordonnancement avec contraintes conjonctives, cumulatives et disjonctives tout à la fois [Roy 1962] , de problèmes de partitionnement avec contraintes additionnelles [Heurgon 1972] , de problèmes linéaires en entiers avec contraintes quelconques [Garfinkel et Nem hauser 1972] , problèmes de colororation sous contraintes [Laurière 1971] .

A ma connaissance, dans la littérature , il n'y a qu'une étude où deux représentations sont considérées et simultanémant utilisées, c'est pour un problème de tournées avec contraintes multiples : [Lemaire 1971] .

La double représentation graphique et algébrique joue ici un rôle non négligeable, le graphe et la partie manipulation formelle s'échangent continuellement des informations selon une procédure qui est maintenant précisée.

II.7.1. Informations du graphe vers la manipulation d'expressions :

Le graphe donne directement les images encore possibles à chaque instant, en particulier les valeurs extrêmes de toute variable sont en permanence accessibles. Les disjonctions interviennent également lors de l'étude algébri-

que, ainsi $\alpha = 0$ ou 1

$$x_1 - x_2 - \alpha = 0$$

et $x_1, x_2 \in \Gamma, \Gamma$ clique de disjonction dans $\mathcal{D}$

$$\Rightarrow \quad \alpha = 1$$

Toute affectation

$$x = \beta$$

donne lieu a une nouvelle règle de réécriture

$$x \rightarrow \beta$$

qui remplace toutes les occurrences de x dans les contraintes algébriques par la valeur numérique β .

Dans ce cas toutes ces contraintes seront de nouveau examinées une à une par le sous programme d'analyse algorithmique décrit en II.4.

En revanche quand le graphe ne fait que supprimer un arc $\vec{ij}$, les contraintes où figurent i ne sont pas réétudiées. Ceci suppose que l'ordre dans lequel on les examine est, en ce sens, optimal.

Cette économie ne peut en aucun cas conduire à une fausse solution, simplement il se peut qu'on n'obtienne pas toutes les déductions possibles du contexte courant; les incohérences éventuelles se rencontreront alors par la suite.

II.7.2. Information de la manipulation formelle vers le graphe :

Il a déjà été dit qu'une nouvelle contrainte telle que

$$x_1 \leq 4$$

est aussitôt transmise au graphe; trois cas peuvent alors se présenter :

- aucune image possible pour x_1 n'est de valeur strictement supérieure à 4 : la contrainte est oubliée.
- toutes les images possibles sont de valeurs supérieures ou égales à 5 : le sous problème ne contient aucune solution et le contexte est oublié.
- certaines valeurs seulement satisfont la contrainte : les autres sont effacées.

Certains problèmes, c'est le cas des problèmes de recouvrement (exemple 5 dans I) , se laissent bien formaliser avec une contrainte de la forme

$$\forall\, i \quad \forall\, j \quad (\; fi = j \Rightarrow \mathcal{P} \;)$$

où $\mathcal{P}$ est une certaine proposition dépendant en général de i et j .

Ainsi dans l'exemple 5 on avait

$$fi = j \Rightarrow \forall\, k \in PRE\ j \quad fk = j$$

Si l'image d'un i est j, alors les images de tous les antécédents de j sont j . Lors d'une affectation quelconque cette contrainte est interprétée et renvoyée au graphe où , bien sûr, les antécédents sont connus et qui procède donc bien à toutes les affectations liées par cette contrainte.

LES ARBORESCENCES

Pour résoudre la plupart des problèmes combinatoires il est nécessaire de faire, puis d'éliminer des hypothèses, de faire des choix et de revenir sur ces choix. En d'autres termes [COOK (1970), KARP(1972)] , on a le plus souvent à faire à des problèmes de type NP-complet : le temps nécessaire à la résolution peut croître plus vite que n'importe quel polynôme fonction de la taille du problème.

Les bons problèmes et les bons algorithmes, ceux dont la complexité est polynomiale,sont en effet fort peu nombreux. Nous connaissons aujourd'hui :

1. les problèmes de distance extrémale dans un graphe (plus court chemin, PERT).
2. le problème de l'arbre minimum.
3. les problèmes de couplage à coût minimum ou de flot maximum.
4. le problème de Johnson (ordonnancement disjonctif avec deux ressources).

Les autres ne sont a priori justifiables que de procédures où il faut faire des choix et donc des "erreurs". Le problème type, auquel bien des problèmes ont maintenant été montrés équivalents, est celui du maximum de satisfaction pour une expression logique sous forme normale disjonctive. Rappelons que la méthode bien connue et très utilisée du simplexe entre dans cette catégorie des algorithmes NP-complets.

Ces procédures où il faut faire des choix ont reçus les noms :

de méthodes arborescentes [FAURE et MALGRANGE 1963]
de branch and bound [LITTLE et al. 1963]
de backtrack [GOLOMB et BAUMERT 1965]
de procédures par évaluation et séparation successives [ROY 1970]
d'algorithmes non déterministes [FLOYD 1968]

Toute la question dans ces procédures arborescentes, auxquelles nous refusons le nom d'algorithme, est de limiter le nombre total des choix.

Il faut éviter l'énumération explicite de l'espace de recherche et arriver au même résultat, en se contentant d'une énumération "implicite" de cet espace.

La méthode pour arriver à cette fin est de faire usage d'heuristiques et de fonctions d'évaluation.

Une heuristique est toute règle de choix qui permet, en moyenne, d'atteindre plus vite la solution que le hasard.

Une fonction d'évaluation est une note attribuée à un sous-problème, qui indique dans quelle mesure ce sous-problème est à même de fournir une solution.

Ces notions ont pour la première fois été utilisées dans des programmes de jeux [SAMUEL 1963] ou de démonstration de theorèmes [PITRAT 1966]

III.1 - DESCRIPTION GENERALE DES PROCEDURES ARBORESCENTES.

L'espace $\mathcal{E}$ dans lequel on recherche la solution d'un problème est partitionné par les choix successifs :

Chaque choix ajoute en fait à l'ensemble initial des contraintes du problème une nouvelle hypothèse (i.e. une nouvelle contrainte).

Nous appellerons préclusion (par opposition à conclusion) tout raisonnement qui, de l'analyse des contraintes initiales et additionnelles dues aux choix effectués, déduit une contradiction et ceci sans avoir fixé la valeur de toutes les inconnues.

Cette démarche généralise en quelque sorte le raisonnement par l'absurde: à chaque étape on considère non deux, mais un nombre fini quelconque d'hypothèses mutuellement exclusives.

Plus vite les contradictions seront détectées par les algorithmes de préclusion - manipulation formelle et analyse du graphe interne - , plus efficace sera la procédure de résolution.

Mais les performances globales, outre les raisonnements algorithmiques décrits au chapitre II , dépendent étroitement des critères qui fixent les choix. La façon même de partitionner $\mathcal{E}$ à chaque choix relève de la métacombinatoire (Chapitre III.2): le programme, pour tout problème, élabore un plan général en construisant des sous-buts. Les choix et le plan lui-même seront très liés à toutes les caractéristiques numériques du problème étudié. Ainsi les traitements effectifs de deux problèmes de même énoncé formel mais de données numériques différentes seront eux-mêmes très dissemblables (IV.3), le programme "s'adaptant" à chaque cas.

III.2 - SOLUTIONS REALISABLES ET DISTANCE A L'OPTIMUM :

On appelle "solution réalisable" d'un problème combinatoire, un ensemble de valeurs des variables qui satisfait toutes les contraintes du problème.

Il peut être plus ou moins facile de mettre en évidence une telle solution et de ce point de vue les problèmes se scindent clairement en deux familles :

- les problèmes pour lesquels il existe une solution triviale,
- les autres.

Des exemples de la première famille sont les problèmes d'affectation (couplage dans un graphe complet), les problèmes de recouvrement (solution $x_i = 1 \quad \forall i$), les programmes mathématiques admettant la solution $x_i = 0, \forall i$.

Des exemples de la seconde :

les problèmes de circuit hamiltonien dans un graphe incomplet,
les problèmes de partitionnement,
les problèmes de coloration avec contraintes,
les problèmes d'ordonnancement en général.

La deuxième famille est de loin la plus nombreuse.

Cette distinction, bien que rarement soulignée dans la littérature, est d'une grande importance pour le programme : si de nombreuses techniques, notamment la réduction des coûts des variables et plus généralement le calcul d'une fonction d'évaluation par défaut, sont utilisables dans les deux cas, il faut dans le second cas, contrairement au premier, ne jamais perdre de vue que le sous-espace étudié après plusieurs choix, peut très bien être vide de solutions: il n'y a pas lieu d'optimiser quand le problème n'a pas plus d'une solution ! Plus généralement pour un problème du second type, le programme n'accordera qu'une attention limitée aux coûts de la fonction économique et une importance plus grande aux "degrés de libertés" résiduels (III.4).

Remarque :

Les problèmes de maximisation se traduisant par changement de signe de la fonctionnelle à optimiser, nous supposerons toujours dans la suite que le programme a ramené le problème à une minimisation .

On appelle "bonne solution" d'un problème d'optimisation toute solution obtenue à l'aide d'une heuristique prenant en compte les coûts de la fonction à optimiser.

Etant données une telle solution correspondant pour la fonction économique Z (à minimiser) à la valeur Z_o, et une évaluation par défaut soit $\underline{Z}$ de Z ($\underline{Z}$ étant par exemple calculée par réductions des coûts) satisfaisant donc à

$$Z^* \geq \underline{Z}$$

où Z^* est la valeur cherchée de Z à l'optimum, et l'on supposera $Z^* \geq 0$, il vient

$$\frac{Z_o - Z^*}{Z^*} \leq \frac{Z_o - \underline{Z}}{\underline{Z}}$$

Moyennant la connaissance de cette fonction d'évaluation par défaut $\underline{Z}$ (cf. III.4) on sait donc contrôler l'erreur relative par rapport à l'optimum.

Pour démontrer que l'optimum est atteint il faut être dans l'une des trois possibilités suivantes :

- On a une solution de valeur minimale :

$$Z_o = \underline{Z}$$

- Le problème initial auquel on adjoint la contrainte

$$Z < Z_o$$

où Z_o est la valeur de la meilleure solution actuelle, est impossible .

- L'espace des solutions a été entièrement étudié.

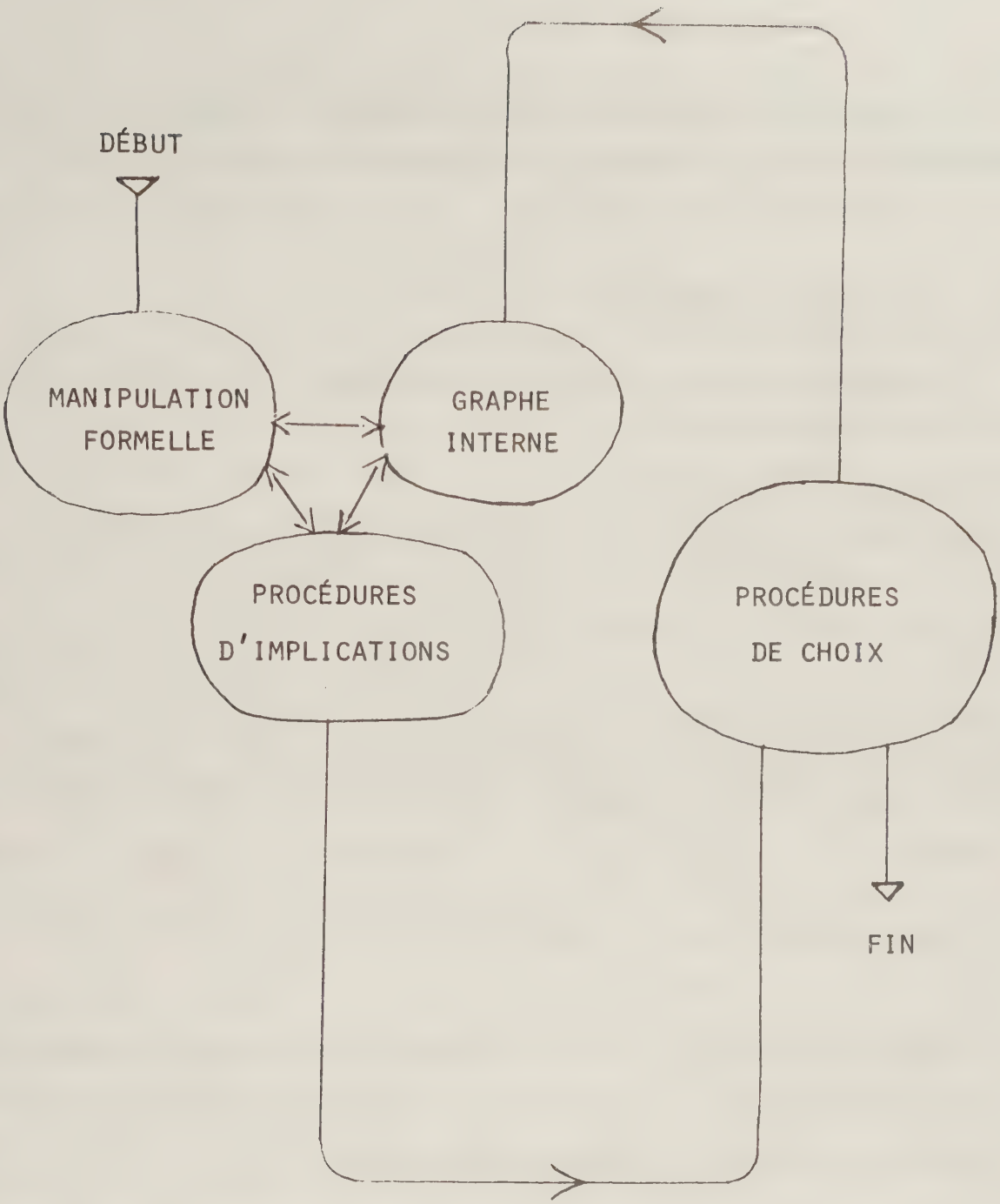
Le programme résout un problème d'optimisation en trois phases :

$\varphi 1$: recherche d'une solution

$\varphi 2$: recherche d'une bonne solution

$\varphi 3$: recherche de l'optimum et démonstration de l'optimalité.

A l'intérieur de chacune de ces phases , il travaille sur les expressions formelles, sur le graphe et effectue des choix suivant le schéma :



DÉMARCHE GÉNÉRALE DU PROGRAMME

À L'INTÉRIEUR DE CHAQUE PHASE

Les traitements algorithmiques des contraintes et du graphe ayant été précédemment décrits, nous donnons dans le paragraphe suivant les différents types de procédures de choix possibles.

III.3 - TROIS TYPES D'ENUMERATION :

Le programme peut se livrer suivant le contexte, en fait, à trois types différents d'énumération.

a) énumération explicite locale :

On entend par énumération explicite l'examen exhaustif de tous les cas. Lorsqu'une contrainte relativement simple est rencontrée par le programme sans que le traitement algorithmique décrit en partie II n'ait pu en déduire de renseignements, le programme examine pour cette seule contrainte toutes les combinaisons possibles (simple signifie nombre de valeurs possibles pour les variables de la contrainte inférieur à un seuil fixé, de l'ordre de quelques dizaines ; ces valeurs sont connues dans le graphe interne).

Exemple :

$$f(2) \neq f(1) - 1 \qquad (1)$$

les images possibles pour 1 étant 6, 7, 8
pour 2 " 3, 4, 9

L'énumération indique que (1) est toujours satisfaite elle est donc détruite après cette procédure.

b) énumération explicite sur l'ensemble du problème.

Il est inutile de se lancer dans des déductions savantes sur un problème de taille faible aussi le programme calcule-t-il, avant de lancer la procédure générale, la dimension D du problème résiduel qu'il a à résoudre.

$$D = \prod_{i \in \mathcal{D}} |\{j \in \mathcal{Q} / \exists \vec{ij}\}|$$

Tenant compte par ailleurs du fait que de toute contrainte en égalité on peut éliminer une variable en l'exprimant en fonction des autres, dès que le rapport D/E , avec E nombre de contraintes en égalité, descend en dessous d'un

seuil s le programme élimine au mieux un sous-ensemble de variables et fait une énumération explicite, ne retenant que les solutions qui satisfont l'ensemble des contraintes algébriques restantes et les contraintes de structures mémorisées dans le graphe interne.

$$D/E < s \Rightarrow \text{énumération}$$

Le seuil s est calculé en partie par le programme lui-même et modifié en cours d'exécution : il dépend de la difficulté a priori (cf. III.5) pour trouver une solution réalisable au problème et est fonction de ce que le programme a déjà fait.

E contient, en fait, outre le nombre de contraintes en égalité, deux termes correctifs pour prendre en compte les autres contraintes :

les contraintes en $\leq$ pouvant donner aisément des informations, tendent à détourner le programme de l'énumération,

les contraintes en ou , en $\neq$ et les autres sont souvent difficilement analysables et dès qu'elles sont en grand nombre par rapport au nombre de variables , sont plus efficacement examinées par énumération et contribuent donc à augmenter la valeur de E . Finalement E est de la forme

$$E = |\{\text{contraintes en } =\}| - \frac{1}{3}|\{\text{contraintes en } \leq\}| + \frac{1}{7}|\{\text{autres contraintes}\}|.$$

Le troisième type est

c) l'énumération implicite qui consiste à faire un choix . C'est-à-dire à ajouter une contrainte additionnelle au contexte , en fixant par exemple la valeur d'une variable, et à en tirer toutes les conclusions avant, si le problème n'est pas résolu, de procéder à un nouveau choix.

C'est cette dernière procédure qui est principalement employée par le programme; elle fait l'objet des paragraphes suivants, traitant successivement des

- choix du choix .
- critères de choix .
- gestion des critères.

III.4 - LES DIFFERENTS PRINCIPES DE SEPARATION : CHOIX DU CHOIX.

Etant donné l'espace $\mathcal{E} = \bigcup_{i \in \mathcal{D}} \{j \in \mathcal{A} , \exists \vec{ij}\}$ à une étape quelconque de la résolution, soit $t = \psi(p)$ où $p = |\mathcal{E}|$, le temps nécessaire pour trouver dans $\mathcal{E}$ une solution qui satisfait toutes les contraintes non exprimées dans le graphe.

Nous savons que $\psi(p)$ est en général non linéaire et même, le plus souvent, plus que polynomiale en p .

En d'autre termes partitionner $\mathcal{E}$ en deux sous-ensembles $\mathcal{E}_1$ et $\mathcal{E}_2$ de cardinalités respectives p_1 et p_2 , avec $p_1 + p_2 = p$, conduira a un temps total de résolution $\psi(p_1) + \psi(p_2)$ qui est inférieur à $\psi(p)$.

Cet argument incline donc à se livrer à des choix et contraindre de plus en plus le problème.

Il existe maintes façons de séparer l'espace de recherche $\mathcal{E}$. Le programme dispose de cinq formes différentes de choix :

La forme de choix la plus simple consiste à attribuer à un certain i de $\mathcal{D}$ une image j dans $\mathcal{A}$,

(1) choix de i puis choix de j et poser $f(i) = j$.

Si tous les j de $\mathcal{A}$ doivent être atteints (surjection), il est inversement possible de choisir un j de $\mathcal{A}$ et de lui associer un antécédent ce qui correspond à

(2) choix de j , choix de i , $f(i) = j$.

Une façon de séparer $\mathcal{E}$ moins finement est de faire

$f(i) \geq \alpha_i$ où α_i est une valeur choisie dans l'intervalle des valeurs possibles pour $f(i)$.

(3) choix de i , choix de α_i , $f(i) \geq \alpha_i$.

Si l'ensemble des arcs $\vec{ij}$ est valué par un coefficient $c(i,j)$ et que le problème est la minimisation d'une fonctionnelle dépendant des $c(i,j)$, les $c(i,j)$ de faibles valeurs seront d'abord retenus :

(4) choix de i , choix de β_i , $\forall i \in \mathcal{D}, \forall j \in \mathcal{A}$ $c(i,j) \leq \beta_i$

β_i étant déterminée de façon à ne retenir qu'un arc $\vec{ij}$ sur deux.

Enfin , un problème où apparaissent une ou plusieurs contraintes commandées par un OU logique (ou inclusif), pourra induire la forme de choix (5) :

(5) choix d'une contrainte p ou q , poser p vraie.

Dans chacun de ces cinq cas, le programme engendre un nouveau contexte, qui satisfait la nouvelle hypothèse, en se ménageant la possibilité de revenir sur ce choix :

Dans les quatre premiers cas la nouvelle hypothèse implique la suppression d'un ensemble $\mathcal{V}$ d'arcs $\vec{ij}$, qui est mémorisé dans une pile. Lorsque le programme revient sur ce choix, il lui suffit de réintroduire l'ensemble $\mathcal{V}$. Dans les cas (1) et (2) il supprimera alors en outre l'arc $\vec{ij}$.

Dans le cinquième cas il ajoutera au contexte courant la condition q vraie , avant de sauvegarder ce contexte.

Tout choix s'effectue donc en deux étapes :

- choix du type de choix,
- choix des paramètres du choix.

CHOIX DU CHOIX :

Chaque fois qu'il en a la possibilité le programme préférera les choix (5) , (3) et (4) dans cet ordre, au choix (1) ou à sa variante (2).

Le choix du choix suit donc un plan algorithmique.

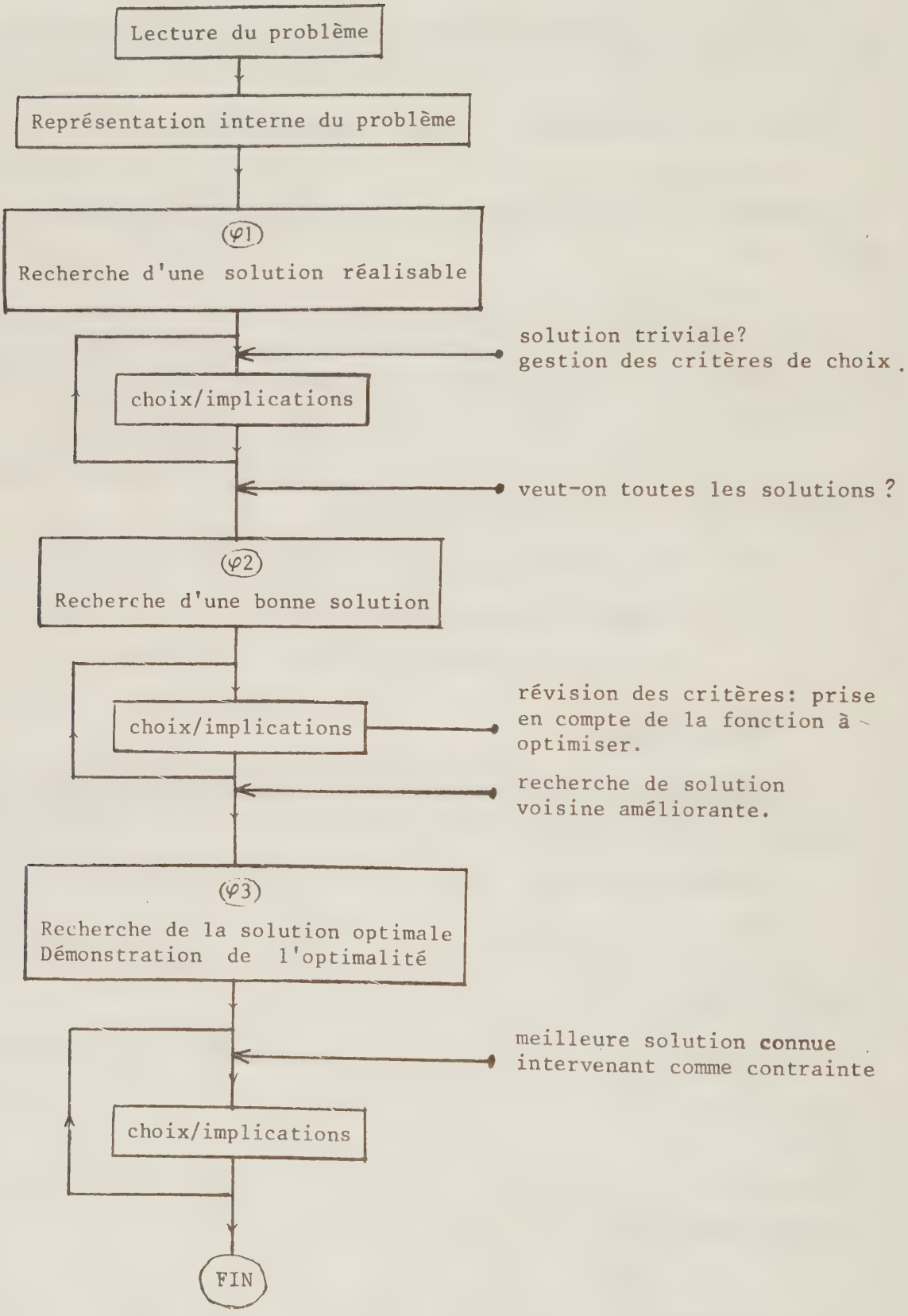
Comme il y a un nombre fini de choix possibles de type (3) (4) et (5) le programme sera amené, s'il n'a pas rencontré d'impasse, aux choix les plus précis de type (1) et (2).

En outre il est imposé au programme de ne faire qu'une fois le type de choix (3) et qu'une fois le type de choix (4) par variable, avant de passer aux choix (1) et (2).

Enfin ces types de choix seront abandonnés au profit de (1) et (2) , pour tout sous problème tel que

$$\min_{i \in \mathcal{D}} |\{j \in \mathcal{A}, \exists \vec{ij}\}| \leq \gamma$$

(γ est un paramètre fixé expérimentalement à 4) et le choix (4) ne peut


Lecture du problème
Représentation interne du problème
(φ1)
Recherche d'une solution réalisable
solution triviale?
gestion des critères de choix.
choix/implications
veut-on toutes les solutions ?
(φ2)
Recherche d'une bonne solution
révision des critères: prise en compte de la fonction à optimiser.
choix/implications
recherche de solution voisine améliorante.
(φ3)
Recherche de la solution optimale
Démonstration de l'optimalité
meilleure solution connue intervenant comme contrainte
choix/implications
FIN

intervenir que dans les phases $\varphi 2$ et $\varphi 3$ de la résolution.

III.5 - CRITERES DE CHOIX .

Les critères retenus pour fixer dans chaque cas les paramètres de choix sont issus de deux "méta"critères simples et qui apparaissent généraux dans ces problèmes combinatoires et/ou d'optimisation.

"méta"-critère 1 : faire le choix le plus informant.

"méta"-critère 2 : faire le choix qui coûte le moins.
(sous-entendu au sens de la fonction économique).

Choix de type (5) : choix d'une contrainte OU pq , p = vraie.

Le programme choisit la contrainte OU de plus petite complexité (cf.II.5) et impose la valeur logique vraie à la proposition de plus petite "complexité" parmi celles commandées par ce OU .

Choix de type (4) : choix de i , choix de β_i , $c(i,j) \leq \beta_i$.

Le sommet i est le sommet, non encore soumis à un choix de ce type pour lequel on a :

$$\max_{i \in \mathcal{D}} \min_{\{j, \exists \vec{ij}\}} c(i,j)$$

et β_i est choisi de façon à ce que

$$|\{j \in \mathcal{A} |\ c(i,j) > \beta_i\}| = |\{j \in \mathcal{A} | c(i,j) \leq \beta_i\}|$$

à une unité près.

Choix du type (3) : choix de i , choix de α_i , $f(i) \leq \alpha_i$.

Le sommet i réalisant

$$\min_{i \in \mathcal{D}} |\{j \in \mathcal{A} | \exists \vec{ij}\}|$$

non encore choisi, est sélectionné. De nouveau α_i est calculé pour séparer équitablement l'espace de recherche.

Choix de type (1) et (2) : $f(i) = j$.

Donnons et expliquons d'abord les critères qui servent aux choix des paramètres de ce type de choix, le plus fréquent.

La façon dont ces critères sont utilisés, et i et j finalement déterminés, dépend

1° de la phase où en est le programme.

2° de l'état d'avancement du sous-problème.

Cette gestion ne sera donc explicite qu'au paragraphe suivant.

CRITERES POSSIBLES DE CHOIX (1) et (2).

A choix de i.

(a) - minimum d'images possibles.
(b) - maximum de sommets en disjonction avec i.
(c) - maximum de contraintes où figure i .
(d) - minimum de la somme des complexités des contraintes où figure i.

(e) - maximum des maximum des coefficients sur i.
(f) - maximum des minimum des valuations des arcs issus de i .
(g) - maximum du "regret".
(h) - maximum d' "intérêt".

B choix de j. (parmi les images possibles de i) :

(α) - minimum d'antécédents potentiels $(|\{i \in \mathcal{D} / \exists \vec{ij}\}|)$
(β) - maximum " "
(γ) - minimum d'antécédents réels $(|\{i \in \mathcal{D} / f(i) = j\}|)$
(δ) - maximum " "
(ε) - minimum de la valeur numérique de j .
(ζ) - maximum " " .
(η) - minimum des coefficients sur j .
(θ) - maximum " "

Le regret d'un sommet i est une notion liée au cas d'optimisation des valuations des arcs $\vec{ij}$; il est égal à l'opposé de la différence entre le coût de l'arc de valuation minimale et celle de l'arc de valuation immédia-

tement supérieure.

Le critère associé conduit à choisir le sommet i et du même coup son image j qui coûterait le plus cher si l'on ne le prenait pas.

L'intérêt d'une variable i est liée à la programmation mathématique : mieux une variable tend à son maximum à satisfaire l'ensemble des contraintes, et plus son coût est faible, plus grand est son intérêt. Cette notion sera mathématiquement définie en §III.7.1.

Enfin si l'on cherche une surjection et s'il existe un sommet arrivée, $j \in \mathcal{A}$, qui a strictement moins d'antécédents que le minimum pour $i \in \mathcal{D}$ du nombre des images potentielles de i, c'est le critère (2) qui sera choisi.

III.6 - ORDRE DE GESTION DES CRITERES.

Au début de chacune des phases $\varphi 1$, $\varphi 2$ et $\varphi 3$, ou chaque fois qu'il a trouvé une nouvelle solution meilleure que les précédentes, le programme ordonne les critères de a à h, et de α à θ, en laissant de côté ceux de ces critères qui, déjà, ne correspondent pas au cas traité : il supprime ainsi de la liste des critères :

(a) - si les inconnues sont booléennes,
(b) - s'il n'y a pas de disjonction ou si tous les sommets sont en disjonction avec le même nombre d'autres sommets,
(c), (d) et (h) - si le problème n'a aucune contrainte algébrique,
(f) et (g) - si il est dans la phase $\varphi 1$, ou si les valuations n'existent pas ou sont toutes égales,
(η) et (θ) - si il n'y a pas de coefficient sur $\mathcal{A}$.

Le critère (a) tend naturellement à décrire l'arborescence la moins large ; il répond au "méta"-critère (1) et sera a priori mis en tête de liste. Cependant il existe un certain nombre de situations où ce critère n'est d'aucune utilité ; le cas des inconnues booléennes a déjà été signalé : toute variable a exactement, et ce tout au long du problème une fois gérées les implications, deux images.

- le nombre minimum d'images possibles est trop grand (par exemple 50) ; il est inutile d'accorder de l'importance au critère avec une telle largeur d'arborescence : il faut trouver autre chose.

- il existe plusieurs sommets, c'est le cas le plus fréquent, qui ont tous le même nombre minimum d'images possibles et il faut les départager.

Les critères suivants, de (a) à (h), sont, pour ce qui concerne $\varphi 1$, ceux de l'ordre alphabétique; ils sont pris en compte dès qu'ils peuvent s'appliquer de la façon suivante :

les critères précédemment rencontrés laissent un sous-ensemble I de sommets ex-aequo ; le meilleur candidat pour le nouveau critère et parmi les seuls i de I, reçoit la note N : on ne retient alors que les i de I qui ont une note supérieure à $N - \left\lfloor \frac{N}{4} \right\rfloor$, les autres sont éliminés. S'il ne reste qu'un candidat, on passe au choix de son image (ou respectivement de son antécédent pour le choix (2)) avec le même schéma de filtrage.

Si le nouvel ensemble I des candidats contient deux éléments au moins, on prend en compte le critère qui suit.

Les critères (a), (b), (c) et (d) correspondant au "méta"-critère 1, il ne sera fait appel à (e), (f), (g) et (h) dans $\varphi 1$ qu'en dernier ressort.

Dans $\varphi 2$, si le programme n'a pas fait pour trouver une première solution plus de $\frac{|\mathcal{D}|}{4}$ retours en arrière dans l'arborescence $\varphi 1$ alors les critères f, g ou h viennent après (a) et (b).

Si le programme a résolu trivialement le problème en $\varphi 1$, ces critères passent en tête dans $\varphi 2$.

Dans $\varphi 3$, on se ramène progressivement à démontrer qu'un problème est impossible; aussi, dès que le programme a effectué plus de $\frac{|\mathcal{D}|}{2}$ retours en arrière dans cette phase, il commence, de nouveau, à rejeter ces critères d'optimisation associés au "méta"-critère 2 (faire ce qui coûte le moins).

Les critères (α) à (θ), présidant au choix des images, poursuivent deux à deux des buts opposés et bien sûr seul un des éléments de chaque couple figurera à la fois dans la liste des critères définitifs.

Dans la phase $\varphi 1$ le critère (α) sera d'abord retenu chaque fois qu'il existe dans $\mathcal{D}$ des cliques de disjonction, (β) quand il n'en existe pas. (γ) viendra alors ensuite si un degré maximum est imposé aux sommets de $\mathcal{A}$, sinon (δ) sera choisi.

Alternativement, à chaque choix, le programme considérera (ε) ou (ζ) en bascule:

cette heuristique qui consiste à "choisir les coins" parait, en moyenne, dans les expériences faites, assez heureuse et bien meilleure en tous cas que le seul hasard. Cette façon de faire reçoit une justification dans des problèmes d'architecture (IV. 5) ou dans certains puzzles (IV. 1).

Le critère (η) interviendra dans tous les problèmes d'optimisation où les coefficients sur les j de $\mathcal{A}$ figurent dans la fonction économique Z, (θ) étant réservé aux problèmes où ces coefficients sont aussi des seconds membres d'inégalité, comme dans le cas

$$\forall j \in \mathcal{A} \qquad \sum_{\{i \in \mathcal{D} \mid f(i)=j\}} a(i) \leq c(j).$$

III.7 - PREMIERE PHASE : RECHERCHE D'UNE SOLUTION REALISABLE.

Sont considérés comme problèmes ayant, pour le programme, une solution réalisable évidente ceux pour lesquels une des trois solutions s_1 , s_i ou $s_\star$ convient.

s_1 : $\forall i$, $f(i) = j_1$ où j_1 est le premier sommet dans la liste des images de i possibles.

s_i : $\forall i$, $f(i) = j_i$ où j_i est le $i^{ème}$ sommet de cette liste.

$s_\star$: $\forall i$, $f(i) = j_\star$ où $j_\star$ est le $(i+1)^{ème}$ sommet.

s_1 correspond à : $\forall i, x_i = 0$ dans un programme mathématique, à l'ensemble vide dans un problème d'ensemble stable de cardinalité maximum.

s_i correspond au choix de la diagonale principale dans un problème d'affectation (IV. 6), à colorer tous les sommets d'une couleur différente dans un coloriage de graphe, à faire $x_i = 1$ dans un programme mathématique en booléens, solution par exemple des problèmes de recouvrements (IV.2).

$s_\star$ correspond au choix du circuit lexicographique dans un problème de voyageur de commerce (IV. 6).

Dans les cas où aucune de ces trois solutions n'est réalisable, le programme ordonne les critères suivant les caractéristiques du problème comme il a été dit précédemment.

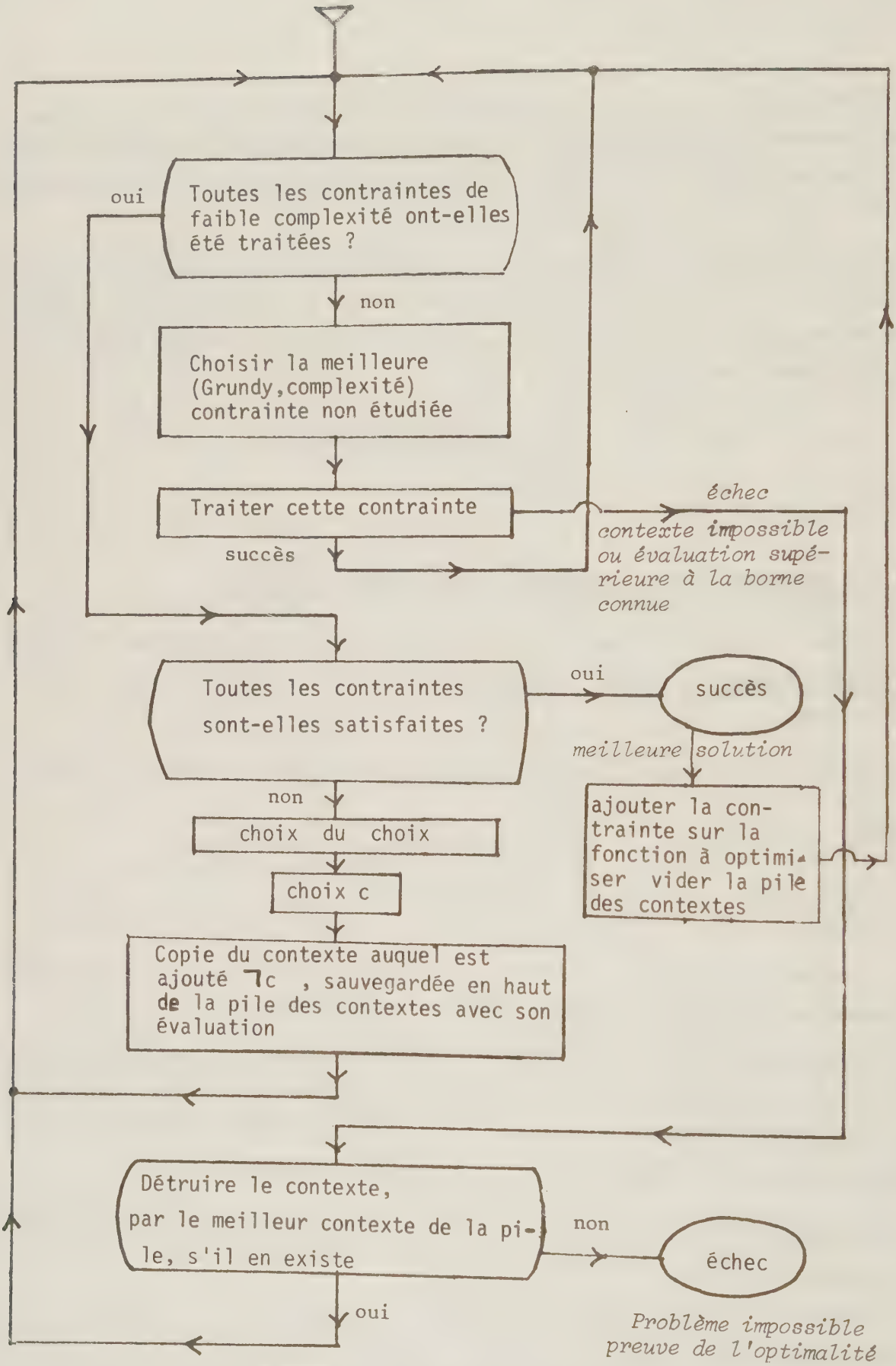


Schéma de fonctionnement des phases $\varphi 1$, $\varphi 2$ et $\varphi 3$.

III.7.1. Recherche d'une solution réalisable entière d'un programme mathématique.

Soit $\{\mathscr{C}_1,\mathscr{C}_2,\ldots,\mathscr{C}_K\}$ l'ensemble des contraintes.

Chaque $\mathscr{C}_k$ s'écrivant sous la forme :

$$\sum_{j\in T^+} t_j - \sum_{j\in T^-} t_j + b \geqq 0$$

où T^+ est l'ensemble des termes positifs,

T^- l'ensemble des termes précédés du signe moins.

Les t_i sont des monômes en fonctions d'inconnues x_i , $i = 1,\ldots,D$ et le problème est de déterminer un ensemble de valeurs entières des x_i qui satisfasse toutes les contraintes. Dans le cas où ils tous du premier degré, l'algorithme du simplexe qui résoud le programme linéaire $\{\mathscr{C}_1,\ldots,\mathscr{C}_K\}$ en supposant les inconnues x_i continues et non plus entières, est utilisé par la quasi-totalité des auteurs comme point de départ : La recherche d'une solution entière au système $\{\mathscr{C}_k\}$ se fait ensuite en examinant les points entiers voisins du point continu donné par le simplexe.

Dans le cas non linéaire le problème est encore très mal résolu à l'heure actuelle , puisque pour aucun programme existant il n'est question de calculer l'optimum entier pour plus d'une quarantaine de variables.

Si toute la difficulté du problème vient de la propriété d'intégrité demandée aux variables, les valeurs possibles de ces variables restent cependant, a priori, de par la nature des problèmes réels qui se laissent modéliser, inférieures à la dizaine; le cas de variables booléennes est même le plus habituel.

Nous distinguerons dorénavant les termes linéaires et les autres dans chaque $\mathscr{C}_k$, qui prend alors la forme :

$$\sum_{\substack{i=1 \\ a_{ik}>0}}^{D} a_{ik}\, x_i + \sum_{\substack{i=1 \\ a_{ik}\leq 0}}^{D} a_{ik}\, x_i + \sum_{M} a_{i_1 i_2 \ldots i_p k}\, x_{i_1} x_{i_2} \ldots x_{i_p} + b_k \geqq 0$$

où la notation $\sum_{M} a_{i_1 i_2 \ldots i_p k}\, x_{i_1} x_{i_2} \ldots x_{i_p}$

est une abréviation désignant la somme de tous monômes M de $\mathscr{C}_k$ non linéaires.

Difficulté d'une contrainte et intérêt d'une variable :

A toute contrainte $\mathscr{C}_k$ sous cette forme nous allons associer un nombre réel d_k appelé difficulté de la $k^{ième}$ contrainte, d'autant plus grande que la contrainte est difficile à satisfaire :

Il est clair que si $b_k \geq 0$, la solution $(\forall i, x_i = 0)$ satisfait trivialement la contrainte, alors la difficulté de celle-ci sera posée égale à 0 :

$$b_k \geq 0 \Leftrightarrow d_k = 0$$

Sinon, soit $$\alpha_k = \sum_{a_{i_1 \cdots i_p k} > 0} \frac{a_{i_1 i_2 \cdots i_p k}}{p^2}$$

la somme des coefficients positifs de tous les monômes divisés par le carré de leur ordre. Cette somme dans le cas des monômes linéaires se réduit à :

$$\alpha_k = \sum_{\substack{i=1 \\ a_{ik}>0}}^{D} a_{ik} .$$

La difficulté de la contrainte k est par définition

$$d_k = \frac{\max(0, -b_k)}{\max(0, b_k + \alpha_k)}$$

Si l'on fait alors dans $\mathscr{C}_k$, $x_j := 1$ il vient

$$(\mathscr{C}_k) \quad \sum_{\substack{i=1 \\ i\neq j \\ a_{ik}>0}}^{D} a_{ik} x_i + \sum_{\substack{i=1 \\ i\neq j \\ a_{ik}\leq 0}}^{D} a_{ik} x_i + \sum_{M \ \forall \ell i_\ell \neq j} a_{i_1 i_2 \cdots i_p k} x_{i_1} x_{i_2} \cdots x_{i_p} + a_{jk} + b_k \geqq 0$$

a_{jk} étant de signe quelconque.

Si $a_{jk} + b_k$ est positif ou nul la difficulté de $\mathscr{C}_k$ lorsque $x_j = 1$, notée $d_k(j)$ est devenue

$$d_k(j) = 0 .$$

Sinon, soit

$$\beta_k = \sum_{\substack{M \\ a_{i_1 i_2 \ldots i_q k} > o}} \frac{a_{i_1 i_2 \ldots i_q k}}{q^2}$$

où l'ordre q de chacun des monômes qui contenait x_j a diminué du degré de x_j dans le monôme. La difficulté est devenue

$$d_k(j) = \frac{\max(0, -b_k - a_{jk})}{\max(0, b_k + a_{jk} + \beta_k)}$$

(Dans le cas linéaire le dénominateur, en fait, n'a pas été modifié puisqu'alors $\alpha_k = a_{jk} + \beta_k$).

L'intérêt, pour ce qui est de la $k^{ème}$ contrainte, de poser $x_j = 1$ est alors d'autant plus grand que la différence $d_k - d_k(j)$ est grande.

Exemple

$$(\mathscr{C}_k) \qquad -x_1 - 2x_3 - 3x_5 + x_1x_4 + 5x_2x_5 - x_4x_5 - 1 \geq 0$$

$$d_k = \frac{1}{-1 + \frac{1}{4} + \frac{5}{4}} \quad \text{soit} \quad d_k = 2$$

$$d_k(1) = \frac{1}{-2 + 1 + \frac{5}{4}} = 8$$

$$d_k(2) = \frac{1}{-1 + \frac{1}{4} + 5} = \frac{4}{17}$$

$$d_k(3) = \frac{3}{\max(0, -3 + \frac{1}{4} + \frac{5}{4})} = \infty$$

$$d_k(4) = \frac{1}{-1 + 1 + \frac{5}{4}} = \frac{4}{5}$$

$$d_k(5) = \frac{4}{-4 + \frac{1}{4} + 5} = \frac{16}{5}$$

Les candidats sont donc pour cette contrainte rangés dans l'ordre x_2 , x_4 , x_5 , x_1 , x_3 .

Cas $d_k = \infty$: Le cas d'un dénominateur nul est traité séparément. Le programme prend alors

$$d_k = b_k + 10^n$$

avec n choisi pour tout le problème tel que n minimum et 10^n supérieur à tous les coefficients.

Cependant, dans le cas linéaire, si le dénominateur de d_k devient nul, cela signifie

si $b_k + \sum_{\substack{i=1 \\ a_{ik} > 0}}^{D} a_{ik} = 0$ que le choix

$x_j = 1$ entraine $x_i = 1 \quad \forall i \quad a_i > 0$

si $b_k + \sum_{\substack{i=1 \\ a_{ik} < 0}}^{D} a_{ik} < 0$ que le choix

$x_j = 1$ conduit à un sous problème impossible et ce choix est interdit.

Intérêt d'une variable.

L'intérêt d'une variable j prend en compte la différence des difficultés d_k et $d_k(j)$ pour, cette fois, l'ensemble des contraintes ; il s'écrit

$$I(j) = \sum_{k=1}^{K} (d_k - d_k(j))*(d_k + 1)$$

III.7.2. Algorithme de résolution d'un programme mathématique en entiers.

Dans $\varphi 1$ le programme ne se préoccupe pas de la fonction à optimiser et se trouve exactement dans le cas du paragraphe précédent :

Les inconnues sont l'une après l'autre fixées à 1 par intérêt maximum. Après chaque choix le sous-programme d'analyse algorithmique (II.4) des seules contraintes modifiées est mis en oeuvre, ainsi éventuellement que l'analyse du graphe et les difficultés sont recalculées, le programme décrivant une arborescence de type général avec le critère (h) (III.4).

Le cas où les inconnues sont entières quelconques (mais cependant bornées) est traité sensiblement de la même façon:

l'inconnue dont l'intérêt est le plus élevé est fixée à son maximum.

III.8 - DEROULEMENT DU PROGRAMME DANS LES PHASES 2 ET 3 .

Le programme parvient au début de $\varphi 2$ en ayant obtenu une première solution réalisable, après R retours en arrière dans l'arborescence. Il ne s'est pas du tout, jusqu'alors, préoccupé de la fonction économique et la valeur Z_o de celle-ci pour la solution obtenue peut être fortement éloignée de l'optimum.

Dans la deuxième phase, la fonction économique est prise en compte.

Cas Z1 : fonctionnelle polynomiale sous contraintes :

Les variables sont rangées dans l'ordre $\gamma(j) * I(j)$

où $$\gamma(j) = \frac{\nu}{\nu - 1 + \sum_{\substack{M^* \\ \exists \ell, j = l_\ell}} (c_{i_1 i_2 \ldots i_p} / p^2)}$$

M^* désignant l'ensemble des monômes de la fonction économique; dans le cas d'une fonction linéaire $Z = \Sigma\, c_i\, x_i$, on a $\gamma(j) = \frac{\nu}{\nu - 1 + c_j}$, ν est un coefficient choisi par le programme et qui donne une influence plus ou moins grande aux coefficients de la fonction économique.

Si le programme a obtenu en φ_1 une solution triviale alors $\nu = 1$.

Sinon $\nu = 10 \cdot R/D$ (R nombre de retours dans l'arborescence φ_1 , D nombre de variables).

L'intérêt d'une variable est d'autant plus élevé que cette variable rend, au moindre coût, l'ensemble des contraintes plus facile à satisfaire.

Dans la phase $\varphi 3$ la contrainte

$$Z < Z_o$$

étant intégrée à l'ensemble des contraintes $\mathscr{C}_k$, on est de nouveau dans le premier cas.

Remarques:

.Dans le cas du knapsack (K=1) , fonctionnelle linéaire et une seule contrainte linéaire

$$\min \Sigma c_j x_j \quad \text{sous} \quad \Sigma a_j x_j \geq b ,$$

on retrouve l'heuristique:

$$\min \frac{c_j}{a_j}$$

.Dans le cas d'un problème de recouvrement

$$\min \Sigma c_j x_j$$

$$\forall k \qquad \sum_{j \in P_k} x_j \geq 1 \qquad P_k \subset \{1,2...D\}$$

on retrouve l'heuristique

$$\min \frac{c_j}{|\{k, j \in P_k\}|}$$

dans ces deux cas le programme pose en effet $\nu = 1$ puisque $(\forall j, x_j = 1)$ est solution.

Cas Z2 : $\min \sum_{i \in \mathcal{D}} c(i,f(i))$

Dans la phase $\varphi 2$ le critère (f)

$$\max_{i=1,\ldots,m} \min_{j=1,\ldots,n} c(i,j)$$

vient en tête de la liste : il tend à faire prendre l'arc le moins cher partant du sommet qui coûte de toutes façons le plus.

Si, en outre, d'un sommet $i \in \mathcal{D}$ on peut atteindre a priori tous les $j \in \mathcal{C}$, une heuristique spéciale décrite ci-après est mise en oeuvre. Le cas fréquent correspond à la donnée d'une matrice rectangulaire de coefficients de laquelle on doit extraire un certain nombre de cases vérifiant une propriété donnée et telle que la somme des coefficients soit minimale : problèmes d'affectation, de transport, de localisation.

L'idée est de mettre toutes les lignes et toutes les colonnes, c'est-à-

dire toutes les inconnues et toutes les images, à égalité, en changeant les c(i,j) par une homothétie. Cette homothétie est construite de façon à rendre nulle sur toute ligne et toute colonne la somme des nouveaux coefficients c'(i,j).

Si :

$$c'(i,j) = \lambda\, c(i,j) + \mu \sum_{i=1}^{m} c(i,j) + \nu \sum_{j=1}^{n} c(i,j) + \xi \ .$$

on veut donc :

$$(\lambda+m\mu) \sum_{i=1}^{m} c(i,j) + \nu \sum_{i=1}^{m} \sum_{j=1}^{n} c(i,j) + m\,\xi = 0$$

$$(\lambda+n\nu) \sum_{j=1}^{n} c(i,j) + \mu \sum_{i=1}^{m} \sum_{j=1}^{n} c(i,j) + n\,\xi = 0$$

Il suffit donc de prendre

$$\lambda = mn \qquad \mu = -n \qquad \nu = -m \qquad \xi = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \ .$$

Cette transformation bien que généralisant les notions connues d'écartements [Berthier 1966] ou d'affinités [Vo Khac Khoan 1969] et [Lemaire 1972] , de différence maximale [Wagner 1969] et conduisant à de bonnes solutions approchées ne semble pas avoir été utilisée dans les travaux antérieurs.

Réductions des valuations des arcs et pénalités :

Dans $\varphi 3$ une nouvelle transformation des coûts est effectuée par le programme . Soit en effet un sommet i relié à des sommets de $\mathcal{A}$ par les arcs de valuations (14,6,8,5,12) . Ce sommet va contribuer dans $Z = \min_{i \in \mathcal{D}} \Sigma\, c(i,j)$ à une augmentation d'au moins 5 .

Le programme réduit donc les coûts en faisant

$$\forall\, i \qquad c'(i,j) = c(i,j) - \min_{j/\exists \vec{ij}} c(i,j)$$

L'ordre des sommets dans lequel le programme procède à ces réductions n'est pas toujours indifférent, il choisira donc un i tel que

$$\min |\{j/\exists\, \vec{ij}\}|$$

et en cas d'égalité

$$\min_{j/\exists \vec{ij}} \min c(i,j) \ .$$

Après ces réductions la fonction objectif s'écrit

$$Z = \underline{Z} + \sum_{i \in \mathcal{D}} c'(i,f(i))$$

avec

$$\underline{Z} = \sum_{i} \min_{j/\exists \vec{ij}} c'(i,j)$$

les $c'(i,j)$ restant par construction positifs, on a :

$$Z \geq \underline{Z}$$

De plus

$$\forall i \quad \exists \vec{ij} \ / \ c'(i,j) = 0 \ .$$

et s'il existe une solution constituée de ces seuls arcs de valuation nulle elle est optimale.

Dans les autres cas et pour de nombreux problèmes comme cela a été souligné dans [Gondran et Laurière (1974) et (1975)] , il est possible de définir le regret (ou la pénalité) d'une variable. Soient un sommet i et j_o un indice tel que $c'(i,j_o) = 0$, le regret $r(i)$ s'écrit :

$$r(i) = \min_{\substack{j/\exists ij \\ j \neq j_o}} c'(i,j)$$

C'est le coût "en second" des arcs issus de i , il est ce que coûte au minimum l'affectation de i , si l'on ne fait pas $f(i) = j_o$.

Le critère (g) $\max_i r(i)$ est alors pris en compte dans $\varphi 3$ pour choisir i et j . On fait $f(i) = j_o$ pour le sommet i qui coûterait le plus cher si $f(i) \neq j_o$.

L'algorithme de Kruskal est ainsi pratiquement retrouvé dans le problème de l'arbre minimum.

Cas Z3 : $\min \max_{i \in \mathcal{D}} f(i)$:

Une première solution de valeur Z_o ayant été obtenue, tous les arcs reliant un sommet i quelconque à un sommet j de valeur supérieure ou égale à Z_o sont définitivement supprimés.

En outre dans $\varphi 2$ le critère ε

$$(\varepsilon) \qquad \min_j \{j \ / \ \exists\ ij\}$$

vient en tête lors du choix d'une image j : $\min \max f(i)$ donnant donc l'heuristique $\min_i f(i)$. Et de plus on essaiera de "resserrer" au mieux la solution par le critère γ

$$(\gamma) \qquad \max_j |\{i \in \mathcal{D} / f(i) = j\}|$$

qui tend à utiliser les images déjà données.

Mais ce critère sera remplacé dans la phase 3 par son opposé δ :

$$(\delta) \qquad \min_j |\{i \in \mathcal{D} / f(i) = j\}|$$

ayant en effet obtenu une bonne solution avec $\max f(i) = Z_o$, il est logique de supposer qu'en construire une de valeur $Z_o - 1$ c'est-à-dire n'utilisant qu'un sous-ensemble de $\mathcal{A}$ de cardinalité $Z_o - 1$ sera chose de moins en moins aisée ; il faut alors au moins songer à faire intervenir les éléments de $\mathcal{A}$ équitablement et leur donner en particulier constamment le même nombre d'antécédents tant que cela est possible, c'est le rôle de (δ) .

Cas Z4 et Z5 :

Le critère ε : $\min \{j/\exists\ ij\}$ est utilisé dans $\varphi 2$ pour le choix de toute image d'une variable qui intervient dans la fonction économique avec un coefficient positif, le critère δ $\max \{j/\exists\ ij\}$ dans le cas contraire.

Dans $\varphi 3$ on ajoute la contrainte

$$Z < Z_o$$

qui se sépare dans le cas Z4 ($\min \max c_i x_i$) en autant de contraintes qu'il existe de variables:

$$\forall\ i \qquad c_i x_i < Z_o$$

La fin de l'optimisation s'obtient dans ces cas, en prouvant que le nouveau problème obtenu n'a aucune solution.

III.9 - RESOLUTION DES CINQ EXEMPLES DE LA PARTIE I .

Les "corrigés" de ces cinq problèmes sont donnés ici dans un but démonstratif pour suivre pas à pas le travail du programme. Ils auraient pu (à l'exception sans doute du second) être résolus par une procédure complètement combinatoire, ce qui est absolument exclu pour les problèmes de dimensions nettement plus importantes présentés dans le chapitre suivant.

Problème 1

$$\min Z = 2x_1x_3 + 4x_2 + 3x_1x_5 + 2x_4 + 3x_5$$

(1) avec $0 \leq x_1x_4 + 3x_2x_5 - x_1 - 2x_3 - x_4x_5 - 1$

(2) $0 \leq 2x_1 + 5x_3 - 2x_4 - 3x_5 - 2$

en variables booléennes.

Dans la première phase, la fonction Z est laissée de côté, le problème n'admet ni la solution $x \equiv 0$, ni la solution $x \equiv 1$, le programme commence donc par chercher une solution réalisable par d'autres moyens et, avant tout, les contraintes (1) et (2) sont analysées.

L'algorithme décrit en II.2 donne ici, à partir de (2) qui est étudiée en premier comme plus simple que (1).

$$0 \leq -2 + 2x_1 + 5x_3$$

elle-même réécrite en

(3) $$x_1 = 1 \quad \text{OU} \quad x_3 = 1$$

Il vient alors à partir de (1)

(4) $$0 \leq -1 + x_1x_4 + 3x_2 \cdot x_5$$

(5) $$0 \leq 3 - x_1 - 2x_3 - x_4 \cdot x_5$$

qui donnent à leur tour

$$0 \leq -1 + 4$$
$$0 \leq 2 + x_1x_4$$
$$0 \leq 3 \cdot x_2 \cdot x_5$$
$$0 \leq 3 + 0$$
$$0 \leq 3 - x_1$$
$$0 \leq 3 - 2 \cdot x_3$$
$$0 \leq 3 - x_4 \cdot x_5$$

et ces sept dernières contraintes sont trivialement satisfaites et donc effacées du contexte.

Il y a alors un choix à faire et comme il existe une contrainte OU , l'arbitrage se fait sur celle-ci et le programme choisit

$$x_1 = 1$$

Il vient

(2) $0 \leq 5x_3 - 2x_4 - 3x_4$

(1) $0 \leq -2 + x_4 + 3x_2x_5 - 2x_3 - x_4x_5$

(1) $\Rightarrow$ $0 \leq -2 + 1 + 3x_2x_5$ normalisée en $x_2 = x_5 = 1$

(1) $0 \leq 1 - 2x_3$ normalisée en $x_3 = 0$

(2) $0 \leq -2x_4 - 3$ qu'il est impossible de satisfaire.

Le programme revient alors sur le premier choix et pose $x_1 = 0$ et donc $x_3 = 1$.

Il vient alors

(1) $0 \leq -3 + 3x_2x_5 - x_4x_5$ et, de nouveau, $x_2 = x_5 = 1$

(2) $0 \leq -2x_4$ soit $x_4 = 0$

et (1) , (4) et (5) sont vérifiées. D'où la solution unique et donc optimale

$$X = (01101) \quad \text{avec} \quad Z^* = 7 ,$$

qui marque la fin de l'exécution en phase $\varphi 1$.

Montrons cependant ce que donne l'heuristique décrite en III.6.1 et qui est utilisée pour construire une "bonne" solution au problème. Nous sommes donc en phase $\varphi 2$ et la fonction économique est prise en compte dans le calcul des intérêts.

La "difficulté" d_1 de la contrainte 1 s'écrit :

$$d_1 = \frac{1}{-1 + \frac{1}{4} + \frac{3}{4}} = 11$$

et

$$d_2 = \frac{2}{-2 + 5 + 2} = 0,40$$

$x_1 = 1 \rightarrow d_1(1) = \dfrac{2}{-2 + 1 + \frac{3}{4}} = 12 \qquad d_1(2) = 0$

$x_2 = 1 \rightarrow d_1(2) = \dfrac{1}{-1 + \frac{1}{4} + 3} = 0{,}44 \qquad d_2(2) = 0{,}40$

$x_3 = 1 \rightarrow d_1(3) = \dfrac{3}{-3 + \frac{1}{4} + \frac{3}{4}} = 13 \qquad d_2(3) = 0$

$x_4 = 1 \rightarrow d_1(4) = \dfrac{1}{-1 + \frac{3}{4}} = 11 \qquad d_2(4) = \dfrac{4}{-4+7} = 1{,}33$

$x_5 = 1 \rightarrow d_1(5) = d_1(2) = 0{,}44 \qquad d_2(5) = \dfrac{5}{-5+2+5} = 2{,}5$

Le calcul des intérêts $I(j) = \sum_{i=1}^{2} [(d_i(j))*(d_i + 1)]$

$I(1) = -1*12 + 0{,}40*1{,}4 < 0$

$I(2) = (11-0{,}44)*12 + 0 = 10{,}5$

$I(3) = -2*12 + 0{,}40*1{,}4 < 0$

$I(4) = 0 - 0{,}93 \times 1{,}40 < 0$

$I(5) = (11-0{,}44)*12+(0{,}40 - 2{,}5)*1{,}4 = 7{,}56$

Seules les variables x_2 et x_5 ont un intérêt positif, en faisant intervenir la fonction économique, on a les intérêts pondérés :

$$I(2) = \frac{10{,}5}{4} \qquad I(5) = \frac{7{,}56}{\frac{3}{4} + 3}$$

et l'on fait donc $x_2 = 1$. En portant dans le système le calcul des intérêts conduit ensuite à faire

$$x_5 = 1$$

il vient alors

$$x_1x_4 - x_1 - 2x_3 - x_4 + 2 \geq 0$$

$$2x_1 + 5x_3 - 2x_4 - 5 \geq 0$$

La difficulté de la première contrainte est nulle, celle de la seconde infinie et comme cette contrainte est linéaire, on a en fait l'implication

$$x_3 = 1$$

La première contrainte restant satisfaite et tous les coûts des variables dans la fonction économique étant positifs, il suffit de faire

$$x_1 = x_4 = 0$$

pour retrouver la solution précédemment obtenue.

PROBLEME 2

```
EX2
***
SCI ENS N    =   INT 1   5
TRC BIJ NAT >-> N   N
        PRF >-> N   N
        CCL >-> N   N
        BCI >-> N   N
        ANI >-> N   N
AVE =   NAT 1   CCL 2
    =   NAT 2   ANI 1
    =   NAT 3   PRF 1
    =   NAT 4   BCI 3
    =   NAT 5   1
    =   CCL 1   BCI 4
    =   CCL 1   +   CCL 5   1
    =   PRF 5   ANI 4
    =   PRF 2   CCL 3
    =   3   BCI 5
    CL  =   NAT 5   -   CCL 4   1   =   NAT 5   +   CCL 4   1
    =   PRF 3   BCI 1
    CL  =   PRF 4   -   ANI 3   1   =   PRF 4   +   ANI 3   1
    CL  =   PRF 2   -   ANI 5   1   =   PRF 2   +   ANI 5   1
FIN
```

Dans le graphe interne, on a

25 sommets à gauche et 25 sommets à droite,
les 5 ensembles d'arrivées isomorphes ayant été recopiés; il y a au départ 5 cliques de disjonctions et a priori tout point possède cinq successeurs, qui sont autant d'images possible.

Les contraintes (5) et (10) imposent déjà deux valeurs parmi les 25 à déterminer :

boi 5 = 3 et

nat 5 = 1 qui donne dans (11) :

col 4 = 0 OU col 4 = 2 et col 4 ne pouvant prendre la valeur 0, il vient immédiatement :

col 4 = 2.

Le programme traite alors la contrainte (1)

$$\text{nat } 1 = \text{col } 2$$

qui impose à nat 1 d'être différent de col 1 , col 3 , col 4 , col 5 et une nouvelle clique de disjonction est ajoutée dans le graphe représentatif et en outre nat $1 \neq 2$ (car col $4 = 2$). De même col 2 est différent de nat 2 , nat 3, nat 4, nat 5, et col $2 \neq 1$.

Les égalités (2),(3),(4),(6),(8),(9),(12) donnent de même naissance à 14 cliques de disjonctions et 10 arcs sont encore supprimés.

Les huit contraintes en égalité ont été complètement traduites et sont définitivement ôtées de l'énoncé algébrique. Il reste

(7) col 1 = col 5 + 1

(13) prf 4 = ani 3-1 OU prf 4 = ani 3+1

(14) prf 2 = ani 5-1 OU prf 2 = ani 5+1

La contrainte (7) est analysée et donne naissance

- aux inégalités :

$0 \leq -1 + 4$ (vraie)
$0 \leq +\text{col } 1 - 2$ (suppression d'un arc)
$0 \leq -\text{col } 5 + 4$ (suppression d'un arc)
$0 \leq +1 + 4$ (vraie)
$0 \leq -\text{col } 1 + 6$ (vraie)
$0 \leq \text{col } 5$ (vraie) ,

- à la règle de réécriture :

$$\text{col } 5 \rightarrow -1 + \text{col } 1 ,$$

- à la suppression de deux nouveaux arcs :

col $5 \neq 5$ car col $1 \neq 6$
col $5 \neq 1$ car col $1 \neq 2$.

Mais la suppression de ce dernier arc ne laisse qu'un arc incident au sommet 1 et implique :

$$\text{col } 3 = 1 .$$

(col $1 \neq 1$ par (7) , col $2 \neq 1$ par (1) , col $7 \neq 1$ par (11)).
Cette dernière affectation impliquant à son tour par le jeu des disjonctions et des degrés minimum :

prf 2 = 1
boi 2 = 1
ani 5 = 2 par (14)

Et seule la contrainte (13) reste alors en lice, le graphe comportant alors 61 arcs sur les 125 de départ, et ne pouvant conclure dans ce contexte, le programme choisit d'examiner l'hypothèse

prf 4 = ani 3 - 1

Ce qui entraine

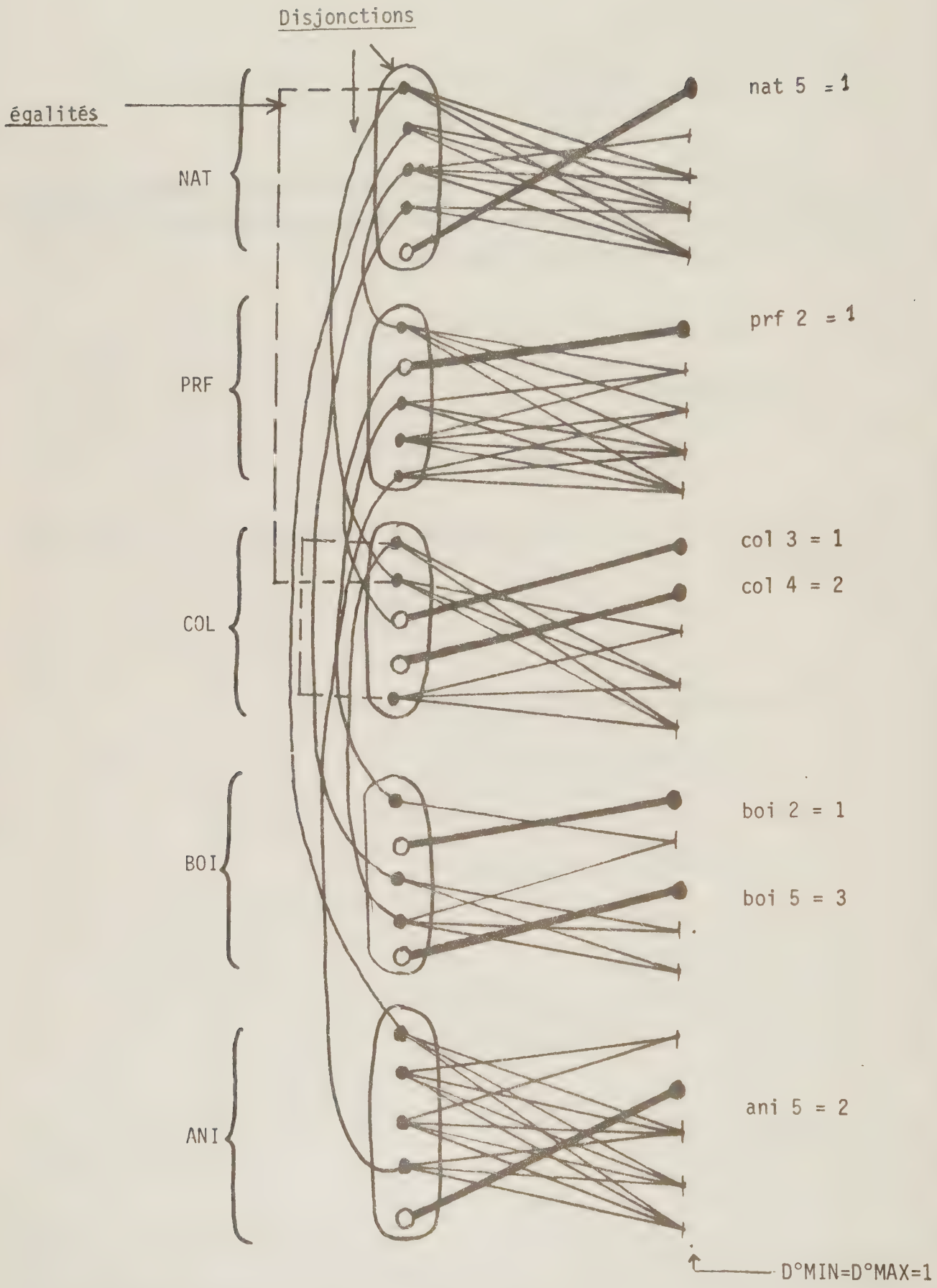
ani 3 ≠ 1 et donc ani 2 = 1

puis ni col 1 = 5 ni col 2 = 5 ne donnent de solution, s'il en existe une c'est donc que

prf 4 = ani 3 + 1 .

Le programme obtient alors l'unique solution :

	1	2	3	4	5
nat	3	4	5	2	1
prf	5	1	4	2	3
col	5	3	1	2	4
boi	4	1	2	5	3
ani	4	5	1	3	2



ETAT DU GRAPHE AVANT LES CHOIX DANS L'EXEMPLE 2

PROBLEME 3 : Emploi du temps.

```
EX3
***
SCI CST N
    ENS S   =   INT 1   N
        F   =   INT 1   N
TRC FCN F   >-> S   F   CIS CMA
AVE MIN MAX F   I
     <   F   5    F   10
     ET  >   F   11   F   4   >   F   11  F   6
FIN
11,
2,3,5,7,8,10,0,0,0,0,0
3,4,7,8,9,11,0,0,0,0,0
5,6,8,0,0,0,0,0,0,0,0,
6,8,10,0,0,0,0,0,0,0,0
7,9,11,0,0,0,0,0,0,0,0
7,10,0,0,0,0,0,0,0,0,0
8,9,0,0,0,0,0,0,0,0,0,
9,11,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,
3,3,3,3,3,3,3,3,3,3,3,
FIN
```

Ce problème possède une solution évidente, peu économique puisqu'elle utilise autant de périodes que de séances : f(i) = i .

Le programme pose alors dans la phase $\varphi 2$, en prenant comme critères :

$$\max_i \left|\{\text{sommets en disjonction avec } i\}\right|$$

d'une part et pour le choix des images :

$$\max_j \left|\{i/f(i) = j\}\right| .$$

Le sous-programme de "garbage collection" , qui compacte les cliques de disjonctions, détecte par ailleurs une clique de quatre sommets : 1,2,7,8.

Suivant les critères rappelés, les choix suivants sont effectués :

$$f(2) = 1$$

$$f(8) = 2$$

f(1) = 3

f(7) = 4

f(3) = 4

f(5) = 1

f(10) = 2

f(6) = 1

f(11) = 4

f(4) = 3

f(9) = 3

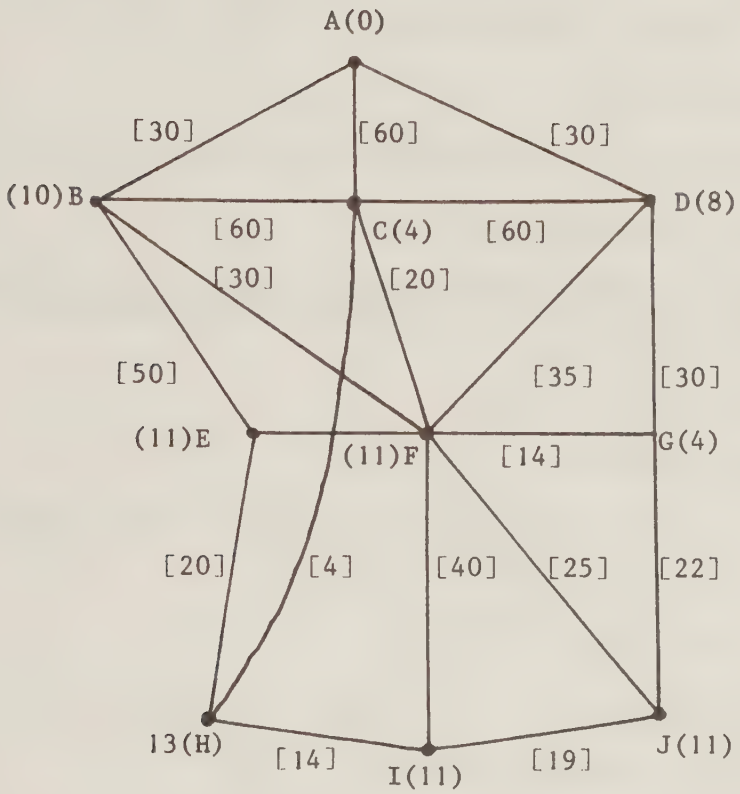
Soit la solution

f(i) =

(i)	1	2	3	4
	(2)	(8)	(1)	(7)
	(5)	(10)	(9)	(3)
	(6)		(4)	(11)

qui, bien qu'aucune affectation forcée n'ait été provoquée, est optimale en vertu de l'existence d'une clique de quatre sommets.

PROBLEME 4 : Soit à trouver un flot non conservatif compatible et arborescent sur le graphe :



Le problème n'admet pas de solution triviale; on fera des choix de type $f(i) = j$ avec les critères

pour i : $\min_i |\{j, \exists \vec{ij}\}|$ 1

$\max_i$ dem (i) 2

pour j : $\max_j$ val (i,j) 3

$\min_j |\{i/f(i) = j\}|$ 4

Les critères 2 et 3 étant dus à la contrainte de compatibilité :

$$\sum_{\substack{i \in \text{Som} \\ \exists \text{ che } ij}} \text{dem (i)} \leq \text{val (j,f(j))}$$

qui est du type décrit en II.6 .

Cette contrainte entraine aussi

$$\forall\, j \in \text{Som}\ ,\ \sum_{i \in \text{Som}} \text{dem}(i) > \text{val}(j, f(j)) \Rightarrow \nexists \text{ che } ij$$

qui conduit à supprimer d'ores et déjà les arcs

$\overrightarrow{EF}$ et $\overrightarrow{FE}$ puisque

$$\text{dem}(E) = \text{dem}(F) = 11 > \text{val}\ (\overleftrightarrow{EF}) = 4\ .$$

Le critère 4 est dû à la recherche d'un arbre .

Le programme choisit alors un sommet parmi ceux qui ont le moins de voisins.

il prend E et F(E) = B

car l'arc $\overrightarrow{EB}$ a la plus grande valuation.

Les arcs $\overrightarrow{EH}$ et $\overrightarrow{BE}$ sont supprimés.

Les sommets G , H , I , J sont alors à égalité pour les deux premiers critères; J est retenu et

$$f(J) = G$$

(F et G sont à égalité).

Les arcs $\overrightarrow{JF}$, $\overrightarrow{JI}$, $\overrightarrow{GJ}$ sont interdits; de plus

$$\text{dem}(J) + \text{dem}(G) = 15 > \text{val}\ (\overrightarrow{GF}) = 14$$

conduit à ôter également $\overrightarrow{GF}$, à l'implication

$$f(G) = D$$

à la suppression enfin de $\overrightarrow{DG}$ par la contrainte "arbre".

Les choix suivants sont

$$f(I) = F$$

$$f(H) = E$$

$$f(B) = C$$

$$f(C) = D \Rightarrow \text{impossibilité}$$

car alors il existe des chemins reliant les sommets

J , G , H , E , B et C à D or

dem(J) + dem(G) + dem(H) + dem(E) + dem(B) + dem(C) + dem(D) = 61

et l'arc $\overrightarrow{DA}$ est supprimé, mais D n'a plus d'image donc

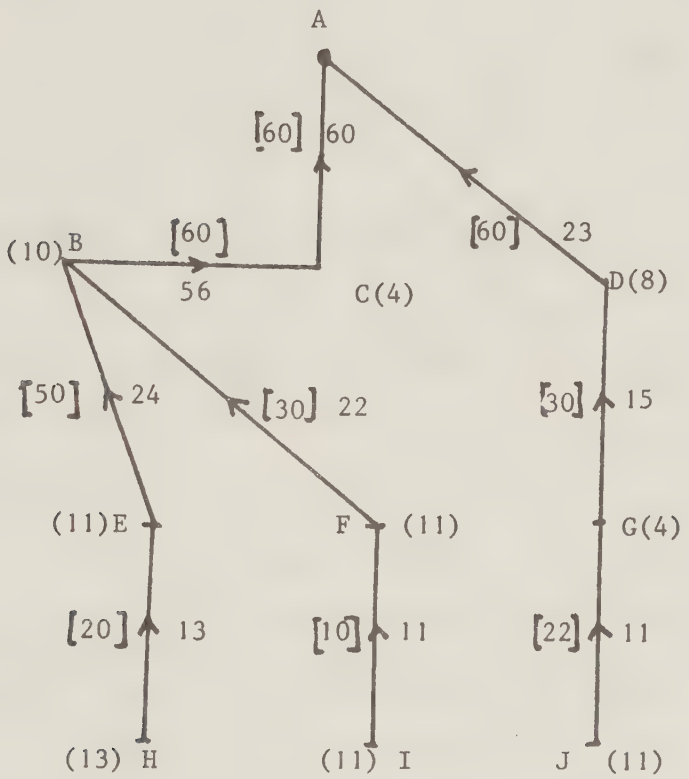
f(C) = A

puis les affectations f(F) = B

f(D) = A

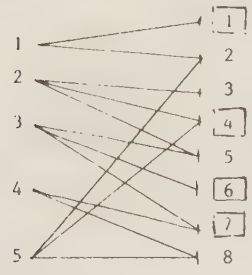
conduisent à la solution :

B	C	D	E	F	G	H	I	J
C	A	A	B	B	D	E	F	G



PROBLEME 5 : Recouvrement optimal :

```
REC
***
SCI CST N   M
    ENS EI  =   INT 1   M
        EJ  =   INT 1   N
    CCE C   SUR EJ
TRC FNL F   >-> EI  EJ  SUC
AVE ==> =   F   I   J   CGS APP L   PRE J   =   F   L   J           ①
    MIN SIG ET  APP J   EJ  EXI APP I   EI  =   F   I   J   C   J   ②
FIN
8,5,
3,7,5,8,10,4,6,9,
1,2,C,
3,4,5,
5,6,7,
7,8,C,
2,4,6,
FIN
```



C	C'			
3	0	0	0	0
7	4	0	0	0
5	5	5	1	1
8	8	4	0	0
10	10	10	6	6
4	4	0	0	0
6	6	6	6	0
9	9	9	9	3

La solution f1 = 1
f2 = 3
f3 = 5
f4 = 7
f5 = 2 convient évidemment ;

la valeur associée de la fonction économique est 3 + 5 + 10 + 6 + 7 = 31 .

Dans $\varphi 2$, le programme cherche une bonne solution : le premier critère

est $\min|\{j/\exists\ \overrightarrow{ij}\}|$ et ② est transposée en un critère habituel :

$$\min_{j/\exists\ ij} C_j \ .$$

Ils donnent lieu aux choix successifs

$f1 = 1$

$f4 = 7 \Rightarrow f3 = 7$ selon ①

$f2 = 3$

$f5 = 6$

avec $Z = 3 + 6 + 5 + 4 = 18$.

Au début de la phase d'optimisation $\varphi 3$, le programme opère la réduction des coûts $c(i)$:

La ligne 1 coûte au moins 3 , d'où $c'(1)=0$ et $c'(2) = 4$;
5 coûte alors au moins 4 et $c'(2)=0$, $c'(4)=4$, $c'(6)=0$;
2 " " 4 et $c'(3)=1$, $c'(4)=0$, $c'(5)=6$;
4 " " 6 et $c'(6)=0$, $c'(9)=3$;

et $\underline{Z} = 3 + 4 + 4 + 6 = 17$

donc la première solution est à 6% au plus de l'optimum

$$17 \leq Z^* \leq 18$$

et dans toute solution meilleure qu'elle, on a $\forall\ i$, $f(i) \neq 3,6,8$ images dont les coûts réduits sont supérieurs à $Z_o - \underline{Z}$.

Un nouveau critère préside au choix de i et de j , c'est

$$\max_i \min_{j/j\neq j^*} c'_j \qquad j^*: \text{indice du premier } c'_j = 0$$

qui correspond au regret maximum.

Puisque l'on a :

i	1	2	3	4	5
$\min_{j,j\neq j^*} c'_j$	0	1	0	3	0

il vient :

$f4 = 7$ et $f3 = 7$, puis

$f2 = 4$ et $f5 = 4$, et enfin

$f1 = 1$, solution dans laquelle tous les c'_j sont nuls, donc

$$Z = \underline{Z}$$

ce qui prouve l'optimalité de cette dernière solution.

LES RESULTATS

TABLE DES RESULTATS

	Nombre de sommets dans $\mathcal{D}$	Nombre de sommets dans $\mathcal{A}$	Nombre de contraintes explicites	Temps CPU en secondes IBM 370/168	Nombre de remontées dans les arborescences
Génération de permutations	9	9	33	4*	0
Exercice 1 programmation en entiers	5	2	3	1	0
Exercice 2 puzzle logique	25	25	14	4	1
Exercice 3 emploi du temps	11	11	6	2	0
Exercice 4 flot arbre	10	10	1	2	0
Exercice 5 recouvrement	5	8	2	3	0
Huit dames	8	8	1	5*	0
—	64	64	2	3*	0
Cinq dames	64	64	1	1	0
Instant insanity	8	12	20	11	3
Informations fausses	10	10	8	1	0
Mike et john au tennis	19	6	21	3	1
Send+more=money	12	18	7	4	0
Crypt addition paramétrée	14	12	9	3	0
Donald+gérald=robert	15	12	9	9	4
Addition parfaite	15	12	8	5	4
See+see+see+yes=easy	4	13	6	2	0
Lyndon*b=johnson	16	14	7	4	0
En toute logique	2	10	2	1	0
Nombres croisés	26	10	28	6	0
Opérations croisées	21	20	37	7	0
Multiplication	29	20	23	249	1826
Carré magique ordre 3	9	9	8	4	0
" " ordre 4	16	16	10	7	0
Carré hétérogène	17	29	1	10	7
Cube ordre 3	27	27	50	18	0
Sphère magique	26	26	8	6	0

* temps pour obtenir la première solution.

	$\|\mathcal{D}\|$	$\|\mathcal{A}\|$	$\|K\|$	T	R
Disques	4	8	9	3	0
Missionnaires et cannibales	80	13	10	11	4
Cavalier d'Euler	64	64	0	5	0
Picnic Fikes	9	6	7	4	0
Couples au tennis	18	18	47	3	0
Allocation de ressources	6	3	2	1	0
	40	10	2	5	0
Partitionnement	15	32	1	4	4
	32	327	1	47	128
	43	327	1	86	628
	45	512	1	91	815
	130	878	1	186	2464
Recouvrement	15	32	1	4	4
	130	878	1	75	429
" bottleneck	45	512	1	8	0
Découpe linéaire	9	9	2	2	1
	54	54	2	3	0
	53	53	2	5	7
	647	70	2	43	953
Choix d'investissement	60	2	30	46**	0
	60	2	30	49**	0
Programmes linéaires	8	2	4	3	1
	10	2	6	2	0
Test Haldi "IBM" n°9	15	35	1	120**	2824*
Programmes quadratiques	8	2	4	4	4
	18	2	4	29	37
	24	2	4	39	98
	32	2	4	92	411
	40	2	4	309	1677
Ordonnancement	100	250	300	26	0
Ordonnancement disjonctif	100	250	310	54	27
Plus court chemin	23	25	56	17	14
Flot maximum	15	65	28	15	16
Affectation	20	20	0	10	18

	|V|	|A|	|K|	T	R
Transport	20	10	20	15	10
Arbre minimum	60	60	0	6	0
Couplage maximum	110	110	0	5	0
Parcours Eulérien	60	60	0	9	5
Découpe dans le plan	24	20	14	4	0
	30	28	22	7	0
Agencement d'un plan au sol	36	26	107	17	0
Voyageur de commerce	6	6	1	1	0
	10	10	1	6	27
	25	25	1	47**	433
	48	48	0	106**	1073
Existence d'un circuit hamiltonien	10	10	0	1	0
	20	20	0	2	0
	64	64	0	5	0
	110	110	0	73	1220
	150	150	0	188	2636
Tournées avec contraintes	10	10	2	13	18
	25	25	2	67	625
	48	48	2	152**	1588
Coloration	40	40	1	17	1
	99	99	1	71	124
	150	130	1	162	250
	40	4	1	6	0
	110	4	1	12	4
	150	4	1	18	24
Planning	186	27	2	16	12
	186	28	2	4	0
Couplage maximal	12	2	2	2	1
	32	2	2	2	0
	80	2	2	241	2056
Papeterie	4	10	2	6	1
	10	30	2	17	14
Localisation	10	10	2	5	0
	20	40	2	30	98
Roulements	40	40	2	17**	24
	40	40	2	14**	47
	53	53	2	31**	138

(** : solutions approchées)

IV.1 - LES PUZZLES.

LE PROBLEME DES HUIT DAMES

Ce problème consiste à placer 8 dames sur un échiquier sans que deux dames quelconques ne soient en prise (une dame prend sur toute case de la ligne, de la colonne, des deux diagonales où elle se trouve). Ce problème, qui revient à trouver un ensemble intérieurement stable de cardinalité maximum sur un graphe de 64 sommets, a intéressé plusieurs mathématiciens au XIX° siècle dont Gauss (qui avait omis des solutions) [Kraitchik 1926].

Le problème revient, pour le programme, à trouver une bijection f de l'ensemble des lignes dans l'ensemble des colonnes avec

$$f\ i = j \Rightarrow \forall\ k \in \text{lignes avec } i \neq k$$

$$f\ k \neq j + k - i \qquad (1)$$

$$f\ k \neq j + i - k \qquad (2)$$

Il est simple de simuler à la main le comportement du programme à l'aide d'un échiquier de 8 dames et de pions de couleur pour gérer la pile des interdictions :

On place de façon systématique à chaque itération une dame sur la rangée, ligne ou colonne, où il reste le moins de places disponibles, on marque de pions de couleurs à raison d'une couleur par itération (choix) les nouvelles cases interdites par (1) et (2); si l'on rencontre un coup forcé ($i^{ème}$ dame en j) , les cases interdites sont colorées de la couleur du dernier choix.

Il est alors aisé de revenir en arrière, soit après avoir trouvé une solution, soit après avoir rencontré une impasse, en ôtant de l'échiquier la(les) dame(s) associées au dernier choix avec les pions de couleurs qui correspondent.

En éliminant,pour raison de symétries les choix :

$$f(1) > 4 \quad \text{et} \quad f(1) = 1 \ ,$$

on est conduit à envisager environ

$$3 \times 5 \times 4 \times 3 = 180 \text{ possibilités.}$$

B	B	(1)	B	B	B	B	B
V	B	B	B	(2)	V	V	V
B	(3)	B	V	B	V	N	N
N	N	B		V	B	V	
	V	B	N	V		B	V
V	N	B		V			B
	N	B		V	N		
	N	B		V		N	

Etat de l'échiquier après les choix (1) associé à B, (2) à V , (3) à N

Dans la plupart des cas les quatre premiers choix imposent les quatre dernières positions. Il ne faut, ainsi, à la main, que quelques heures pour obtenir 42 solutions, qui se ramènent après éliminations des solutions encore équivalentes par symétrie et rotation aux douze solutions non isomorphes.

24683175	35281746 - (symétrique)
25713864	35286471
25741863	35841726
26174835	36258174
26831475	36428571
27368514	36815724

```
SCI CST N
    ENS LIG =   INT 1   N
        CCL =   INT 1   N
TRC BIJ F   >-> LIG CCL
AVE ==> =   F   I   J   CCS APP K   LIC ==> ¬=  1   K   ET  -=  C   +   I   +
MNS K   +   MNS J   F   K   ¬=  F   K   +   J   -   I   K
AVE ET  <=  F   1   /   N   2   ¬=  1   F   1
FIN
8.
```

Il existe une deuxième formulation, plus simple, de ce problème. Soit G_{64} le graphe symétrique à 64 sommets, où un sommet est une case de l'échiquier et où il existe une arête xy dès que x et y sont sur une même rangée ou une même diagonale : il s'agit maintenant de trouver l'ensemble

intérieurement stable maximum dans G_{64} [Berge 1967] , c'est-à-dire aussi une injection de l'intervalle [1,64] dans [1,8] qui respecte les disjonctions de G_{64} .

Le programme trouve toutes les solutions, par les mêmes heuristiques, mais sans avoir à interpréter continuellement les contraintes (1) et (2) contenues dans le graphe représentatif interne : il y gagne un rapport de 9 dans le temps d'exécution (on passe de 195 à 22 secondes).

Le graphe G_{64} serait un peu long à entrer à la main, mais il y a bien sûr été obtenu dans une première passe par le programme lui-même à partir de (1) et (2).

PROBLEME DES CINQ DAMES :

Il faut placer cinq dames sur l'échiquier de façon à ce que toute case soit contrôlée par une au moins de ces cinq dames.

En se servant du même graphe G_{64} le problème est de construire maintenant un stable extérieur minimum, soit une fonction multivoque de [1,64] dans lui-même, avec

$$j \in f(x) \quad \Rightarrow \forall y, \exists \vec{xy} \text{ dans } G_{64} \quad j \in f(y)$$

Le critère principal retenu par le programme est ici de choisir le j pour lequel on a

$$\max |\{i / \exists \vec{ij}\}|$$

c'est-à-dire de choisir la case qui contrôle le plus de cases non encore couvertes.

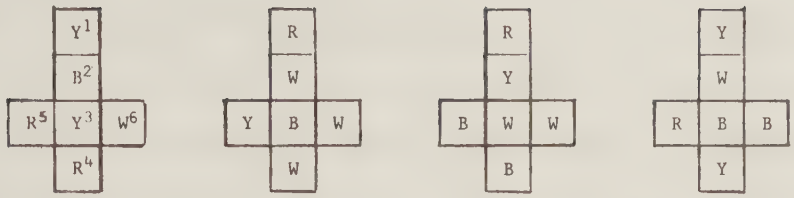
Le problème est d'ailleurs un problème de recouvrement minimal cf. exercice 5 .

INSTANT INSANITY

Ce puzzle est vendu sous ce nom dans le commerce.

Il s'agit d'empiler quatre cubes, dont les faces sont colorées, de telle façon que chaque face du parallélépipède obtenu présente exactement quatre

couleurs.



(1) (2) (3) (4)

```
INSTANT INSANITY
****************
SCI ENS CCL =   CBJ
        N3  =   INT 1   3
        N4  =   INT 1   4
        N6  =   INT 1   6
    CCE CPP SLR N6
    CCA CLB >-> PCT N4  N6  CCU
TRC FCN FA  >-> N4  N6
        GA  >-> N4  N6
AVE CGS APP I   N4  ET  ¬=  FA  I   GA  I   ¬=  CPP FA  I   GA  I
AVE CGS APP I   N3  CGS APP K   INT 4   I   1   4   ET  ¬=  CLB I   FA  I   CLE
K   FA  K   ET  ¬=  CUB I   GA  I   CUB K   GA  K   ET  ¬=  CLB I   CPP FA  I
CLB K   CPP FA  K   ¬=  CUE I   CPP GA  I   CUB K   CPP GA  K
FIN
R   B   W   Y
3 4 1 2 6 5
Y   B   Y   R   R   W
R   W   B   W   Y   W
R   Y   W   B   B   W
Y   W   B   Y   R   B
FIN
```

Chaque cube est repéré par sa face avant (FA) et sa face gauche (GA).

Le vecteur OPP donne dans un cube la face opposée à chaque autre face.

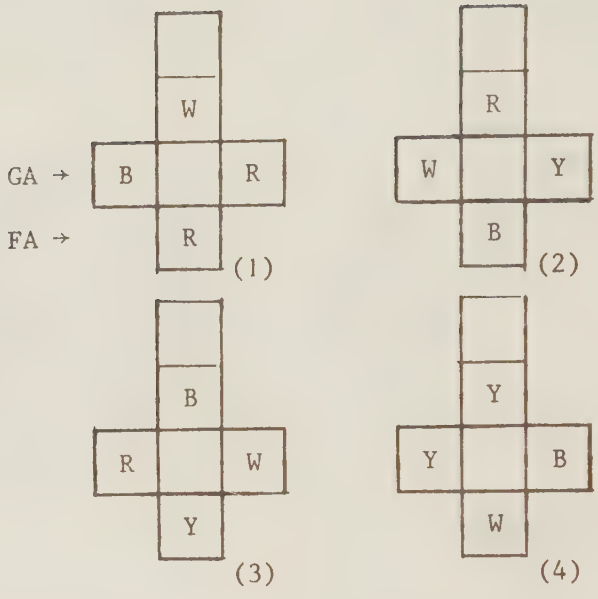
Les critères retenus correspondent à

- la variable qui figure dans le plus de contraintes
- la couleur qui apparait le plus souvent

d'où les choix successifs.

```
        ga(2)=6
          ↑
        ga(4)=1
⇒       ga(3)=1
⇒       ga(1)=2
          ↑
        fa(4)=2
⇒       fa(1)=5
⇒       fa(2)=3
⇒       fa(2)=2
```

solution :



Le programme fait grand usage de la règle

$$v(i,x) \neg = k \Rightarrow x \neq j \quad \text{si} \quad v(i,j) = k$$

dans laquelle i et k sont connus et x inconnu.

De nombreux auteurs, tels Busacker et Saaty [1968] , se sont intéressés à ce problème et ont donné différents modèles en termes de graphes.

Le programme résout ici la question de façon satisfaisante puisque les quatre solutions équivalentes par rotation de 90 degrés du parallélépide sont obtenues en 11 secondes; un autre exemple où le problème est impossible est étudié en 14 secondes.

UN PUZZLE LOGIQUE AVEC INFORMATIONS FAUSSES :

H. Muller-Merbach [1973] propose le problème :

Dans un certain état a eu lieu un scrutin parlementaire d'une grande importance. Alors que le parti α l'a emporté de justesse sur le parti β , certains journalistes écrivent que Mr. I (Incorruptible), membre du parti β , a voté α , après que le parti α lui ait offert une confortable somme d'argent. Une enquête est ouverte et les parlementaires α qui,en bons parlementaires, mentent un fois sur deux déposent :

Mr A dit : Aucun d'entre nous cinq n'est le trésorier des fonds secrets du parti; mais je sais que Mr I a reçu l'argent de Mr B qui le tenait de Mr D via Mr C .

Mr B dit : L'argent a été reçu par Mr I . Mr C est le trésorier.

Mr C dit : La deuxième assertion de A est vraie. La deuxième de B également.

Mr D dit : Mr I n'a jamais reçu d'argent de notre parti. Je suis bien placé pour le savoir puisque je suis trésorier du parti.

Mr E dit : Mr D n'a jamais été trésorier. J'ai reçu l'argent de Mr B et l'ait moi-même donné à Mr I .

L'une des propositions de chacun des parlementaires est donc vraie, l'autre fausse. Ce puzzle est donc une version un peu plus complexe de ceux plus classiques de Lewis Carroll

Une formulation possible est

```
SCI ENS N   =    INT 0   1
        M   =    INT 1   2
IRC BIJ A   >-> M   N
        B   >-> M   N
        C   >-> M   N
        D   >-> M   N
        E   >-> M   N
AVE >=  1   +    A   1   B   2
    >=  1   +    A   1   D   2
    >=  1   +    D   2   E   1
    >=  1   +    B   2   D   2
    >=  1   +    D   1   B   1
    =   1   +    A   2   B   2
    =   A   2    D   1
    =   D   2    B   2
FIN
```

La valeur 0 correspondant à faux et 1 à vrai , cinq contraintes du type

$$A1 + A2 = 1$$

sont adjointes automatiquement. Le programme travaille alors de la façon suivante :

d'après ce que dit C , soit il est le trésorier et I n'a rien reçu, soit I a reçu l'argent de D et l'un des cinq est le trésorier .

((6) B1 = 1 - A1 et B2 = A1 : la première phrase de A et la seconde

de B sont soient vraies, soient fausses simultanément).

Mais la première assertion de A et la seconde de B "aucun n'est trésorier" et "C'est le trésorier" sont contradictoires ((1) A1 + B2 ≤ 1) donc la première de A est fausse (2 A1 < 1) et la solution A2, B1, C1, D2, E2 sont vraies.

MIKE ET JOHN AU TENNIS (Fikes)

Mike a été battu par John au tennis 6 jeux à 3 .

Cinq fois, celui qui était au service a perdu son jeu. Qui a servi le premier ?

```
MIKE JOHN AU TENNIS
*******************
SOI ENS JEU =    INT 1   9
        BOO =    INT 0   1
    COE SER SUR JEU
TRO CST S
    FON MG   >-> JEU BOO
        JG   >-> JEU BOO
AVE =   5   SIG APP I   JEU +   *   MOD +   S   I   2   JG  I   *   +   1   MNS
MOD +   S   I   2   MG  I
AVE =   6   SIG APP I   JEU JG  I
    QQS APP I   JEU =   JG  I   -   1   MG  I
    OU  =   S   0   =   S   1
    =   JG  9   1
```

Le choix S = 0 donne par élimination des JG(i) = 1 - MG(i) :

$$5 - [MG(1)+MG(3)+MG(5)+MG(7)] + MG(2)+MG(4)+MG(6)+MG(8) = 5 \qquad (1)$$

$$9 - [MG(1)+MG(2)+MG(3)+MG(4)+MG(5)+MG(6)+MG(7)+MG(8)] = 6 \qquad (2)$$

d'où par combinaison (élimination de MG(1)) :

$$2[MG(2) + MG(4) + MG(6) + MG(8)] = 3 \quad \text{impossible.}$$

S = 1 donne alors pareillement :

$$4 - [MG(2)+MG(4)+MG(6)+MG(8)] + MG(1)+MG(3)+MG(5)+MG(7) = 5 \qquad (1)$$

(2) est inchangée donc cette fois :

$$[2\ MG(1) + MG(3) + MG(6) + MG(7)] = 4$$

qui donne une famille de solution, c'est donc John qui a servi le premier.

Couples au tennis : Nous reprenons l'énoncé en langage naturel et en REF de la thèse de Fikes.

English Statement: Six couples took part in a tennis match. Their names were Howard, Kress, McLean, Randolph, Lewis, and Rust. The first names of their wives were Margaret, Susan, Laura, Diana, Grace and Virginia. Each of the ladies hailed from a different city; Fort Worth Texas, Wichita Kansas, Mt. Vernon New York, Boston Mass., Dayton Ohio, Kansas City Mo.; and finally, each of the women had a different hair color, namely black, brown, gray, red, auburn, and blond.

The following pairs played doubles: Howard and Kress against Grace and Susan, McLean and Randolph against Laura and Susan. The ladies with black and brown hair played first against Howard and McLean, then against Randolph and Kress. The following singles were played: Grace against McLean, Randolph and Lewis, the gray haired lady against Margaret, Diana and Virginia, the lady from Kansas City against Margaret, Laura and Diana, Margaret against the ladies with auburn and blond hair, the lady from Wichita against Howard and McLean, Kress against Laura and Virginia, the lady from Mt. Vernon against the ladies with red and black hair, McLean against Diana and Virginia, and the girl from Boston against the lady with gray hair.

Finally, Rust played against the lady with black hair, the auburn girl against Diana, Lewis against the girl from Kansas City, the lady from Mt. Vernon against Laura, the one from Kansas City against the auburn one, the woman from Wichita against Virginia, Randolph against the girl from Mt. Vernon, and the lady from Fort Worth against the redhead.

There is only one other fact we ought to know to be able to find the last names, home towns, and hair colors of all six wives, and that is the fact that no married couple ever took part in the same game.

REF Statement:

```
begin;
    set HOWARD of HUSBAND to select(1,6);   | set BLOND of HAIR to select(1,6);
    set KRESS of HUSBAND to select(1,6);    | set RED of HAIR to select(1,6);
    set MCLEAN of HUSBAND to select(1,6);   | set AUBURN of HAIR to select(1,6);
    set RANDY of HUSBAND to select(1,6);    | set FWT of HOME to select(1,6);
    set LEWIS of HUSBAND to select(1,6);    | set KMO of HOME to select(1,6);
    set RUST of HUSBAND to select(1,6);     | set WK of HOME to select(1,6);
    set BLACK of HAIR to select(1,6);       | set MVNY of HOME to select(1,6);
    set BROWN of HAIR to select(1,6);       | set BM of HOME to select(1,6);
    set GREY of HAIR to select(1,6);        | set DAT of HOME to select(1,6);
    set vector L to HOWARD,HOWARD,KRESS,KRESS,MCLEAN,MCLEAN,
        RANDY,RANDY,HOWARD,HOWARD,MCLEAN,MCLEAN,RANDY,RANDY,
        KRESS,KRESS,MCLEAN,MARGE,DIANA,MARGE,LAURA,MARGE,MARGE,
        HOWARD,MCLEAN,KRESS,KRESS,MVNY,MVNY,MCLEAN,MCLEAN,BM,
        RUST,DIANA,LEWIS,LARUA,KMO,VIRGIN,RANDY,FWT,RANDY,LEWIS;
    set vector R to GRACE,SUSAN,GRACE,SUSAN,LAURA,SUSAN,LAURA,
        SUSAN,BLACK,BROWN,BLACK,BROWN,BLACK,BROWN,BLACK,BROWN,
        GRACE,GREY,VIRGIN,KMC,DIANA,AUBURN,BLOND,WK,WK,LAURA,
        VIRGIN,RED,BLACK,DIANA,VIRGIN,GREY,BLACK,AUBURN,KMO,
        MVNY,AUBURN,WK,MVNY,RED,GRACE,GRACE:
    set MARGE of NAME to 1;
    set SUSAN of NAME to 2;
    set LAURA of NAME to 3;
    set DIANA of NAME to 4;
    set GRACE of NAME to 5;
    set VIRGIN of NAME to 6;
```

```
        condition excl(HOWARD of HUSBAND, KRESS of HUSBAND, RUST of
            HUSBAND, LEWIS of HUSBAND, MCLEAN of HUSBAND, RANDY of
            HUSBAND);
        condition excl(BLACK of HAIR, RED of HAIR, BLOND of HAIR,
            BROWN of HAIR, AUBURN of HAIR, GREY of HAIR);
        condition excl(FWT of HOME, KMO of HOME, BM of HOME, MVNY of
            HOME, DAT of HOME, WK of HOME);
        for I → 42 doto LAST;
        set <T> to L[<I>];
        set <U> to R[<I>];
        if <T> = HOWARD ∨ <T> = KRESS ∨ <T> = RUST ∨ <T> = LEWIS ∨
            <T> = RANDY ∨ <T> = MCLEAN then MAN;
        if <T> = MARGE ∨ <T> = VIRGIN ∨ <T> = GRACE ∨ <T> = DIANA ∨
            <T> = LAURA ∨ <T> = SUSAN then WOMAN;
CITY.HAIR:condition ~ (<T> of HOME = <U> of HAIR);
        goto LAST;
    MAN:if <U> = MARGE ∨ <U> = VIRGIN ∨ <U> = SUSAN ∨ <U> = LAURA ∨
            <U> = GRACE ∨ <U> = DIANA then MAN.WOMAN;
        if <U> = BLACK ∨ <U> = BROWN ∨ <U> = BLOND ∨ <U> = RED ∨
            <U> = GREY ∨ <U> = AUBURN then MAN.HAIR;
MAN.CITY:condition ~ (<T> of HUSBAND = <U> of HOME);
        goto LAST;
MAN.WOMAN:condition ~ (<T> of HUSBAND = <U> of NAME);
        goto LAST;
MAN.HAIR:condition ~ (<T> of HUSBAND = <U> of HAIR);
        goto LAST;
  WOMAN:if <U> = RED ∨ <U> = BLOND ∨ <U> = BROWN ∨ <U> = BLACK ∨
            <U> = GREY ∨ <U> = AUBURN then WOMAN.HAIR;
```

```
WOMAN.CITY:condition ~ (<T> of NAME = <U> of HOME);
           goto LAST;
WOMAN.HAIR:condition ~ (<T> of NAME = <U> of HAIR);
           LAST:;
end;
```

Ce puzzle est résolu en trois secondes (IBM 370/168) , sans aucun choix, par implications successives. C'était le problème le plus important pour Fikes [7mn d'exécution (IBM 360/67) et 340 instructions interprétées] .

```
IEN
***
SCI ENS E   =   INT 1   6
TRC BIJ NCM >-> E   E
        CCL >-> E   E
        VIL >-> E   E
AVE ET  ¬=  NCM 1   5   ¬=  NCM 1   2
    ET  ¬=  NCM 2   5   ¬=  NCM 2   2
    ET  ¬=  NCM 3   3   ¬=  NCM 3   2
    ET  ¬=  NCM 4   3   ¬=  NCM 4   2
    ET  ¬=  CCL 1   NCM 1   ¬=  COL 1   NCM 3
    ET  ¬=  CCL 2   NCM 1   ¬=  COL 2   NOM 3
    ET  ¬=  CCL 1   NCM 4   ¬=  CCL 1   NCM 2
    ET  ¬=  CCL 2   NCM 4   ¬=  CCL 2   NCM 2
    ET  ¬=  5   NCM 3   ET  ¬=  5   NCM 4   ¬=  5   NCM 5
    ET  ¬=  CCL 3   4   ET  ¬=  COL 3   1   ¬=  CCL 3   6
    ET  ¬=  VIL 6   1   ET  ¬=  VIL 6   3   ¬=  VIL 6   4
    ET  ¬=  1   CCL 5   ¬=  1   CCL 6
    ET  ¬=  VIL 2   NCM 1   ¬=  VIL 2   NCM 3
    ET  ¬=  NCM 2   3   ¬=  NCM 2   6
    ET  ¬=  VIL 3   CCL 4   ¬=  VIL 3   CCL 1
    ET  ¬=  NCM 3   4   ¬=  NCM 3   6
    ¬=  VIL 4   CCL 3
    ¬=  NCM 6   CCL 1
    ¬=  CCL 5   4
    ¬=  NCM 5   VIL 6
    ¬=  VIL 3   3
    ¬=  VIL 6   CCL 5
    ¬=  VIL 2   6
    ¬=  NCM 4   VIL 3
    ¬=  VIL 1   CCL 4
FIN
```

PROBLEMES DE CRYPT OPERATION :

I - SEND + MORE = MONEY :

```
SMM
***
SCI ENS LET =    INT 1   8
        RET =    INT 1   4
        CHI =    INT 0   9
        BCC =    INT 0   9
TRC INJ F        LET CHI
    FCN C        RET BCC
AVE =   +   F    4   F   2   +   F   8   *   1C  G   1
    =   +   G    1   +   F   3   F   7   +   F   2   *   1C  G   2
    =   +   G    2   +   F   2   F   6   +   F   3   *   1C  G   3
    =   +   G    3   +   F   1   F   5   +   F   6   *   1C  G   4
    =   G   4    F   5
    -=  F   5    0
    -=  C   F    1
```

Dans cet énoncé le programme ne sait pas au départ que les retenues ne peuvent valoir que 0 ou 1 .

L'examen d'une contrainte telle que (1)

$$f4 + f2 - (f8 + 10.g1) = 0$$

$$\underbrace{[0,9]}\,\underbrace{[0,9]} \quad \underbrace{[0,9]} \quad [0,90]$$

$$[P,PP] = [0,17] \qquad [0,99] = [N,NN] \qquad \text{(cf II.4)}$$

entraîne la nouvelle condition

$$f8 + 10.g1 \leq 17$$

d'où de nouveau $f8 \leq 17$ vraie

et $10.g1 \leq 17$ soit $\underline{g1 \leq 1}$

et le même résultat pour toutes les retenues

(5) donne alors avec (6) : $g4 = f5 = 1$ (8)

qui portée dans (4) permet d'écrire

$$g3 + f1 = f6 + 9 \quad \text{entrainant} \quad f6 \leq 1$$

donc avec (8) $f6 = 0$ et $f1 = 9 - g3$

(3) $g2 + f2 = f3 + 10.g3$
$1,10 \qquad 1,19$

(3) $\Rightarrow$ $f3 + 10.g3 \leq 10 \Rightarrow g3 = 0$ donc $f1 = 9$
$1,9 \quad 0,10$
$f2 \neq f3 \Rightarrow g2 = 1$ et $f3 = f2 + 1$

(2) $g1 + f7 = 9 \Rightarrow g1 = 1$ et $f7 = 8$

Il reste (1) $f4 + f2 = f8 + 10$ et $f3 = f2 + 1$.

Donc $4 \leq f2 \leq 7$ et $f4 \geq 5$

et l'unique solution :

$f2 = 5 \quad f3 = 6 \quad f4 = 7 \quad f8 = 2$

PARAMETRISATION

L'utilisation des quantificateurs et du symbole Σ permet d'écrire l'énoncé de toute la famille de problèmes de crypt addition :

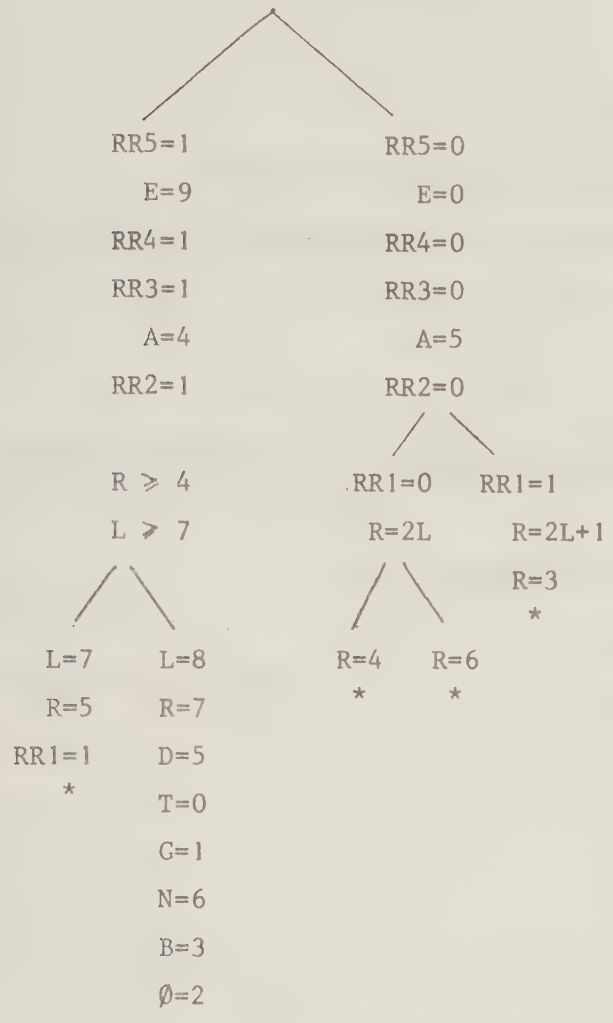
```
PARA CRYPT ADDITION
*******************
SOI CST NL  NC
    ENS LIG =   INT 1   NL                          ensemble  des lignes
        COL =   INT 1   NC                              "     colonnes
        LIN =   INT 1   -   NL  1                     ------------------
        LET =   CEJ                                     "     lettres
        CHI =   INT 0   9                               "     chiffres
        CHT =   INT 0   -   NL  2                     ------------------
        RET =   INT 1   NC                              retenues
    COA AA  >-> FCT LIG COL LET
TRC INJ FF  >-> LET CHI
    FIN RR  >-> RET CHT
AVE CCS APP JJ  COL =   +   SIC APP II  LIN FF  AA  II  JJ  RR  JJ  +   FF  AA
NL  JJ  *   10  RR  -   JJ  1
AVE =   C   FF  0
    ¬=  C   FF  AA  NL  1
    =   C   RR  C
    =   RR  NC  0
FIN
3,5,
S   E   N   D   M   O   R   Y
C   S   E   N   D
C   M   O   R   E
M   O   N   E   Y
FIN
A   B   C   D   E   F   G   H   I   J
C   C   A   B   A
C   B   C   L   E
F   C   H   I   J
FIN
```

54321 ← retenues

ARBORESCENCE DECRITE PAR LE PROGRAMME POUR LE PROBLEME

```
  DONALD
+ GERALD
--------
  ROBERT
```

Les RRi sont les retenues .



ADDITION PARFAITE : Il n'existe qu'une seule addition qui, à la fois utilise le maximum de lettres différentes (dix), ne permet aucun échange de la valeur des lettres, comporte un nombre minimum de symboles (douze) et présente une unique solution. Elle a été mise en évidence par M.DUBOUE c'est ABA + BCDE = FGHIJ.

Il existe de nombreux puzzles de même nature.

Citons les additions :

see + see + see + yes = easy

adam+and+eve+ on + a = raft

forty + ten + ten = sixty

send + more + gold = money

les multiplications :

xa * x = bxc

two * two = three

lyndon * b = johnson

le puzzle (Le Monde du 9/4/75, En toute Logique, P.Berloquin)

"Une personne a en 1898, un âge égal à la somme des chiffres de l'année de sa naissance".

98 - 10 * D - U = 1 + 8 + D + U

(solution unique D=7 , U=6).

Quelques variantes, un peu plus sophistiquées, ont encore été soumises au programme :

Nombres croisés parus dans le Petit Archimède n°10 .

Opérations croisées (Hör zu juillet 72)

```
ABCD  :  EFG  =  HC
 -       *       +
FHA   -   IG  =  FJD
 =       =       =
AFBE  -  ACFE =  FAJ
```

Suivant le même principe (injection lettres → chiffres), le nombre de contraintes augmentant, la tâche du programme en est facilitée.

Utilisant une fonction de Grundy, le programme montre que f(J)=0 puis

f(A) f(H) f(I) avec de plus

f(I) f(E) , f(I) f(F) , f(I) f(C)

et f(A) f(D) , f(A) f(B) , f(A) f(G)

d'où f(A) = 6 , f(I) = 4 , f(H) = 5 , il obtient la solution sans construire d'arborescence.

Multiplication magique

Etant donné le "format" d'une multiplication en système décimal

trouver cette opération de façon à utiliser deux fois chacun des chiffres de 0 à 9 (DMI ≡ DMA ≡ 2) .

La solution (179 × 224) est unique !

Elle demande à la main une bonne journée d'efforts, le programme l'obtient et démontre l'unicité en 4 minutes 10 .

Carrés magiques

Carré magique d'ordre trois:

```
MSP
***
SCI ENS CHI =    INT 1   9
        CAR =    INT 1   9
TRC BIJ F    >-> CAR CHI
    CST SCM
AVE =   SCM +    +   F   1   F   2   F   3
    =   SCM +    +   F   4   F   5   F   6
    =   SCM +    +   F   7   F   8   F   9
    =   SCM +    +   F   1   F   4   F   7
    =   SCM +    +   F   2   F   5   F   8
    =   SCM +    +   F   3   F   6   F   9
    =   SCM +    +   F   1   F   5   F   9
    =   SCM +    +   F   3   F   5   F   7
FIN   16
```

Le programme ajoute à l'ensemble des contraintes

$$\sum_{i \in CHI} f\,i = 45 \qquad (9)$$

car seul l'opérateur + apparait dans les contraintes et qu'il doit trouver une bijection.

Il élimine ensuite f2 qui est la première des variables les moins

contraintes (elle apparait dans 3 équations) :

(1) ⇒ f2 → som + mns f1 + mns f3
et reporte sa valeur dans (5) et (9) . De même

(4) ⇒ f1 → som + mns f4 + mns f7

(5) ⇒ f3 → mns som + f4 + f5 + f7 + f8

(2) ⇒ f6 → som + mns f4 + mns f5

(6) ⇒ f4 → f5 + mns f7 + f9

(3) ⇒ f7 → som + mns f8 + mns f9

La contrainte (8) s'écrit à ce stade

0 = mns som + 3 f 5 soit som = 3.f5

la contrainte (6) disparait (système dépendant), la contrainte (9) est devenue : som = 15 soit f5 = 5 .

La fin ne présente aucune difficulté.

Carré hétérogène

Berloquin propose la construction de carrés dans lesquels toutes les sommes en lignes, colonnes et diagonales doivent être différentes. Il suffit en langage ALICE d'ajouter 8 inconnues g(i) i=1 à 8 égales aux sommes cherchées, g devant être injective pour obtenir la solution

5 4 3
6 1 2
7 8 9

après, cette fois, 5 choix dans l'arborescence.

Carré magique d'ordre quatre :

L'élimination de variables ne permet pas ici, et ceci est général aux carrés d'ordre pair, de fixer la valeur d'une case centrale comme cela est possible pour tout carré d'ordre impair.

Le choix de f1 = 1 est effectué et le programme conclut par énumération explicite pour la solution

1	14	12	4
12	9	6	7
5	8	11	10
16	3	2	13

Cube magique d'ordre trois

Il s'agit cette fois de placer les 27 premiers entiers dans un cube 3×3×3. Chaque sous ensemble de neuf nombres dans un même plan doit avoir la même somme. Le programme ajoute

$$\sum_{i=1}^{27} f(i) = 126 \qquad (16)$$

Et par élimination trouve la valeur de la variable centrale, la plus contrainte:

$$f(14) = \frac{som}{9}$$

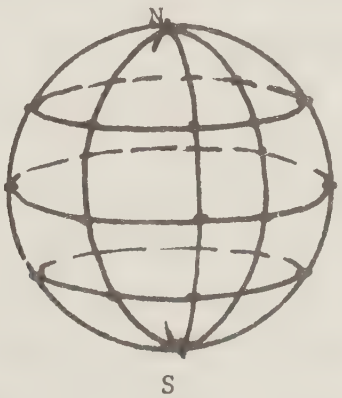
et la valeur de som en portant dans (16)

$$som = 14 \qquad f(14) = 14$$

puis la solution.

Sphère magique :

Placer les 26 premiers entiers aux intersections des cercles sur la sphère:



Les sommes sur chaque cercle doivent être égales.

Le programme obtient de la même façon

$$Som = 108 = 4[f(N) + f(S)]$$

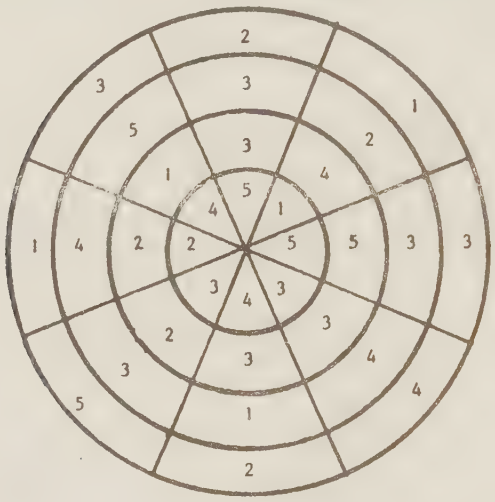
$$27 = f(N) + f(S)$$ et une solution, parmi les nombreuses

possibles, s'obtient alors sans aucune difficulté.

Les disques de BANERJI [1972]

Dans ce problème bien que le programme obtienne très vite la réponse, la démarche est moins satisfaisante.

Soient quatre cercles concentriques, divisés chacun en huit zones valuées:



Il s'agit de faire tourner les disques de façon à amener dans chacun des huit octants la même valeur totale.

```
SCI ENS D   =   INT 1   4
        Q   =   INT 1   8
        C   =   PRC C   C
    MAT V   SUR C
IRC FCN R   >-> C   Q
AVC =   R   1   8                                                    ①
    CCS APP J   C   =   12  SIC APP I   C   V   I   +   1   MCC 8   -   J   +
R   I   1                                                            ②
FIN
2,1,3,4,2,5,1,3,
3,2,3,4,1,3,4,5,
3,4,5,3,3,2,2,1,
5,1,5,3,4,3,2,4,
```

Le disque extérieur reste dans la position initiale(①).

La contrainte (1) donne huit équations. La première d'entre elles s'écrit:

$$10 = v(2,1+(-r(2))_8) + v(3,1+(-r(3))_8) + v(4,1+(-r(4))_8) .$$

Le programme, lors du premier choix, considère la contrainte qui a "le plus grand second membre", il prend la septième. Il fait alors en sorte que $v(2,1-(6+r(3))_8)$ ait la plus grande valeur possible, soit 5 pour r(2) = 7

r(1) = 8 2 1 3 4 2 5 1 3
r(2) = 7 2 3 4 1 3 4 5 3

La plus petite somme est 4 , d'où le choix (deuxième contrainte) :

r(3) = 6 ⇒ r(4) = 8 ne convient pas.
r(3) = 7 ⇒ r(4) = 4 donne la solution.

Simulation d'une machine de Turing.

Partant d'un certain état initial, dans un univers donné où un nombre fini d'actions sont possibles, on cherche à atteindre un état final connu. Un automate non déterministe permet alors de résoudre le problème . Une des actions possibles est choisie à chaque pas, avec retour au choix précédent lorsqu'un état déjà rencontré est réobtenu et arrêt quand on parvient à l'état final. Un exemple classique où le problème est naturellement posé sous cette forme est celui des missionnaires et des cannibales :

N missionnaires et autant de cannibales sont d'un même côté d'une rivière; ils disposent d'un bateau qui supporte P passagers.

Comment transporter tout le monde de l'autre côté en faisant en sorte qu'à aucun moment, sur aucune des rives pas plus que sur le bateau les missionnaires soient inférieurs en nombre aux cannibales ?

Application avec N=3 et P=2 :

```
MC
**
SCI CST N   L   M
    CAS NCF =   INT 1   L
        NMC =   INT 0   N
        BCC =   INT 0   1
        PAS =   INT 0   M
IRC FCN CFE >-> PAS NCF                          ← choix de l'action.
        MIS >-> PAS NMC
        CAN >-> PAS NMC                          ↓— description des actions possibles.
        BAR >-> PAS BCC
AVE ==> =   CFE I   1   ET  =   BAR I   -   1   BAR -   I   1   ET  =   MIS I
MIS -   I   1   =   CAN I   +   CAN -   I   1   -   1   *   2   BAR I
AVE ==> =   CFE I   2   ET  =   BAR I   -   1   BAR -   I   1   ET  =   MIS I
+   MIS -   I   1   -   1   *   2   BAR I   =   CAN I   CAN -   I   1
AVE ==> =   CFE I   3   ET  =   BAR I   -   1   BAR -   I   1   ET  =   MIS I
+   MIS -   I   1   -   2   *   4   BAR I   =   CAN I   CAN -   I   1
AVE ==> =   CFE I   4   ET  =   BAR I   -   1   BAR -   I   1   ET  =   MIS I
MIS -   I   1   =   CAN 1   +   CAN -   I   1   -   2   *   4   BAR I
AVE ==> =   CFE I   5   ET  =   BAR I   -   1   BAR -   I   1   ET  =   MIS I
+   MIS -   I   1   -   1   *   2   BAR I   =   CAN I   +   CAN -   I   1   -
1   *   2   BAR I
AVE CCS APP I   INT 1   M   CCS APP J   INT C   -   I   1   CL  ¬=  BAR I   BAR
J   CL  ¬=  MIS I   MIS J   ¬=  CAN I   CAN J
AVE ==> =   0   +   MIS I   +   CAN I   BAR I   FIN
AVE CCS APP I   PAS ET  CU  =   C   MIS I   <=  CAN I   MIS I   CL  =   N   MIS
I   <=  CAN I   MIS I                                   ← état final
AVE ET  =   1   BAR 0   ET  =   3   MIS C   =   3   CAN C       ← état initial
FIN
3,5,5
FIN
```

Ce problème du même type que le problème du singe et des bananes, célèbre en intelligence artificielle, ou que le problème de transvasement de récipients [POLYA 1962][FIKES 1968] a été jugé digne d'intérêt par de nombreux auteurs [AMAREL 1966]

En fait il n'y a guère mieux à faire que du combinatoire, ce qui marche d'ailleurs fort bien. Le seul souci du programme est de ne pas "boucler" (contrainte (α)).

A chaque étape, cinq actions sont a priori possibles, correspondants aux 5 valeurs de la fonction "ope". La solution, marquée par (β) mis i = can i = bat i = 0 , est atteinte en 11 étapes, par exemple :

$$\left.\begin{array}{l} 331 \xrightarrow{5} 220 \xrightarrow{2} 321 \xrightarrow{4} 300 \xrightarrow{1} 311 \xrightarrow{3} 110 \\ 000 \longrightarrow 111 \longrightarrow 101 \longrightarrow 031 \longrightarrow 020 \longrightarrow 221 \end{array}\right] 5$$

où $ijk \xrightarrow{\ell}$ indique qu'il y a i missionnaires, j cannibales sur la rive de

départ avec le bateau si et seulement si k=1 , et qu'on fait alors l'action ℓ (ope = ℓ)

Dans tous ces problèmes, les choix sont, bien sûr, faits sur la variable dont dépendent toutes les autres ici ope . L'heuristique (min affd) consistant à prendre pour l'opérateur la valeur la moins souvent utilisée, ce qui suppose que toutes les actions possibles sont également utiles, donne dans ce cas de bons résultats.

Le cavalier d'Euler

```
SCI CST N
    ENS SCN =    INT 1    N
TRC EIJ F    >-> SCN SCN SUC SYM CIR
FIN
64,0,C,C,0
11,18,C,C,
12,17,19,0
9,13,18,20
```

Il faut faire parcourir à un cavalier tout l'échiquier avec la marche habituelle aux échecs. L'énoncé formel tient en trois lignes dès que l'on se donne les cases d'arrivée possibles à partir d'une case donnée. Les données numériques ont été coupées après la troisième ligne.

Les critères sont :

case d'où il y a le moins de départs possibles

case où il y a le moins d'arrivées possibles.

Les premiers choix sont les cases étant numérotées de gauche à droite et de haut en bas

f64 = 54 ⇒ f47 = 64

f57 = 51 ⇒ f42 = 57

f 8 = 14 ⇒ f23 = 8

f 1 = 18 ⇒ f11 = 1

f63 = 48

f48 = 38

f46 = 63

f30 = 40 etc...

Le programme aboutit sans aucun retour en arrière à une solution et sept solutions sont obtenues en sept secondes. Par exemple, les cases étant cette fois numérotées dans l'ordre dans lequel elles sont parcourues depuis la première à gauche :

1	4	33	60	63	18	35	20
32	59	64	3	34	21	62	17
5	2	31	58	61	16	19	36
30	57	6	15	50	37	22	45
25	14	29	10	23	44	51	38
56	7	24	49	28	41	46	43
13	26	9	54	11	48	39	52
8	55	12	27	40	53	42	47

IV.2 - PROGRAMMATION MATHEMATIQUE EN NOMBRES ENTIERS.

De très nombreux auteurs ont proposé das méthodes pour résoudre les problèmes combinatoires posés en termes d'inconnues entières et d'équations sur les inconnues. La plupart d'entre eux partent d'une résolution continue du problème et de l'algorithme du simplexe. Le programme travaille ici sans cet algorithme. Si l'on savait résoudre par un algorithme général des problèmes correspondants à des données réelles (quelques milliers de variables et de contraintes) , la programmation mathématique serait un bon outil pour les problèmes combinatoires. En fait, nous en sommes loin et les codes existants ne sont pas fiables, en ce sens que leur temps de calcul peut varier de façon considérable: un programme qui donne "en moyenne" de bons résultats peut s'avérer inutilisable pour le même problème et des données semblables [Fulkerson, Nemhauser Trotter (1974)]

Je pense, personnellement, que pour ce qui est des problèmes concrets, issus du réel , et de taille importante, la recherche d'un optimum est tout à fait illusoire et qu'une solution approchée, donnant, si possible, la distance à cet optimum entier inconnu, est économiquement plus satisfaisante; le simplexe coûte, en autre, le plus souvent, trop cher en temps d'exécution.

Pour ce qui est d'autres problème combinatoires, issus par exemple de la théorie des graphes, et qui sont souvent mis sous forme de programmes mathématiques, je dis que c'est là une mauvaise façon de procéder et qu'il faut garder pour ces problèmes leur formulation en terme de graphes et d'ensembles.

Dans les paragraphes suivants sont successivement donnés les résultats concernants :

- les problèmes de partitionnement et de recouvrement,
- les problèmes de type "choix d'investissements",
- d'autres problèmes linéaires,
- des problèmes quadratiques.

Les chapitres IV.2 à IV.9 traitent de problèmes combinatoires qui ne sont pas formulés en programmes mathématiques.

IV.2.1. Problèmes de partitionnement et recouvrement.

Les modèles de partitionnement et de recouvrement ont été jusqu'à présent très utilisés en recherche opérationnelle pour traduire des problèmes combinatoires réels tels :

tournées [Vo Khac Khoan 1970], rotations d'équipage [Agard et al 1967] [Thiriez 1969] , habillages ou roulements d'horaire [Heurgon 1972] , découpage électoraux.

Ils sont caractérisés par la formulation :

$$(PP) \qquad \min \sum_{i=1}^{n} c_j x_j \qquad \text{avec } x_j = 0,1$$

$$\text{et} \qquad AX \geqq 1 \qquad \geq \text{ recouvrement} \quad = \text{ partitionnement}$$

où A est une matrice booléenne m×n donnée, X un vecteur n×1 d'inconnues booléennes de coûts $c_j > 0$.

On aboutit à ce modèle par une démarche, à mon sens, artificielle et peu satisfaisante : L'étude d'un modèle concret s'effectue en deux temps : un générateur fabrique les inconnues x_j et conduit à (PP) , puis un algorithme spécifique résoud ce nouveau problème.

Les m lignes de la matrice A à laquelle on aboutit dans la phase de génération peuvent être regardées comme l'ensemble des services à assurer. Etant donnés, d'autre part, les moyens (machines ou personnel) qui peuvent assurer ces services, le générateur organise ces moyens de (presque) toutes les façons possibles en satisfaisant toutes les conditions statuaires de travail : temps de repos, durée moyenne de travail, respect de contraintes dues à l'éloignement du personnel de son lieu d'habitation etc... Le générateur est donc un crible qui ne retient que les seuls éléments de solution (i.e. inconnues x_j) acceptable. A chaque x_j est alors associé un coût correspondant aux heures supplémentaires, à la distance parcourue ou à diverses pénalités.

A ce stade, un service donné peut donc être assuré ("habillé") par plusieurs x_j différents : on s'est donc ramené à un problème à choix multiples de partitionnement ou de recouvrement suivant qu'une ou plusieurs équipes peuvent participer à un même service (équipage surnuméraire dans les avions, parcours haut

le pied dans les chemins de fer).

Cette démarche "générale" comporte deux points faibles :

1°) le générateur doit tronquer sévèrement et arbitrairement les solutions possibles, sinon leur nombre trop grand empêche de pouvoir traiter le problème en deuxième phase. On se contente, en général, d'un millier d'inconnues, alors qu'il faudrait en considérer des centaines de mille. La notion rigoureuse d'optimum est donc abandonnée à ce stade.

2°) Alors que les problèmes de partitionnement ou recouvrement, auxquels arrivent les différentes entreprises à travers le monde, ont tous les mêmes caractéristiques moyennes : taille, valeur des coefficients, densité de la matrice, la méthode de la compagnie α n'est pas cessible à la compagnie β, car elle donne alors un temps de calcul rédhibitoire : les problèmes posés dans chaque entreprise, après la phase générateur, sont en effet encore plus structurés que ne le laisse apparaître (PP) et les informaticiens ne se sont pas fait faute, après souvent plusieurs années de travail, d'utiliser toutes les particularités de leur problème pour obtenir une solution. Encore doivent-ils se contenter d'une solution entière à ε% de la solution optimale. Il y a là un échec grave des méthodes de recherche opérationnelle.

Nous ne prolongerons pas plus la discussion dans ce paragraphe et donnons les résultats obtenus sur le problème (PP) une fois posé, mais montrerons en IV.4 comment les problèmes réels d'horaires peuvent être abordés différemment.

Le programme distingue nettement partitionnement et recouvrement puisque dans le second cas il obtient d'emblée la solution triviale $X \equiv 1$, (Cf. exercice 5), qui n'est pas solution dans le premier cas.

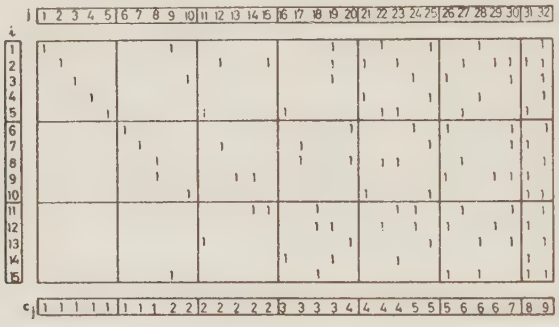
Par ailleurs, si les problèmes peuvent effectivement être posés sous la forme (PP), il est clair, qu'en ALICE, la formulation, déjà donnée, (Cf. exercice 5) utilisant la notion d'ensemble pouvant saturer une ligne (successeurs) et sa réciproque (antécédents d'une colonne) permet de mémoriser A sous une présentation plus accessible par le programme de résolution et conduit donc à des temps de calcul beaucoup plus courts (gain d'un facteur 10 sur l'exemple 15×32 présenté dans la suite).

Enoncé du problème de partitionnement :

```
SCI CST N   M
    ENS LIC =   NN  1   M
        CCL =   NN  1   N
TRC FCN F   DEA LIC CCL SUC VAL
AVE ==> =   F   I   J   CQS APP L   PRE J   =   F   L   J                    (1)
    ==> =   F   I   J   CQS APP L   SUC I   ==> ¬=  L   J   CQS APP K   PRE L
                                                                            (2)
==> ¬=  K   I   ¬=  F   K   J
AVE =   F   1   2
FIN
```

C'est la contrainte (2) qui rend la recherche de solutions réalisables plus difficile dans le problème de partitionnement.

Exemple [Lemke, Salkin, Spielperg 1971]



- Solution réalisable pour le problème de partitionnement.

La ligne qui a le moins de successeurs est la ligne 14 .

Les colonnes 19 et 23 couvrent cette ligne et possèdent le maximum d'antécédents, le programme pose

$$f(14) = 23 \qquad \text{(i.e } x_{23} = 1 \text{ dans la formulation (PP))}$$

d'où $f(2), f(5), f(8), f(11) = 23$ par (1)

et $f(7) \neq 12, f(1) \neq 19, f(3) \neq 19, f(12) \neq 19$ etc... par (2).

La ligne 12 n'a plus alors qu'un successeur possible et donc

$$f(12) = 26$$

d'où $f(3), f(6), f(9), f(15) = 26$ par (1)

et $f(1), f(4), f(7), f(10), f(13) = 25$ par (2)

et la solution, obtenue en première phase, de valeur 14.

Dans la seconde phase, le critère en tête pour les recouvrements correspond à $\min_i \min_{j/\exists \vec{ij}} c_j$, il sera pour les partitionnements, précédé par celui du "nombre de successeurs minimum".

On trouve ainsi une solution de valeur 18 pour le recouvrement, la même solution que précédemment pour le partitionnement (la branche $f(14) = 19$ ayant en effet conduit à une impossibilité).

Les mêmes critères seront conservés dans la dernière phase, mais il y a au préalable une réduction des coûts.

Le programme calcule

$\underline{Z} = 13$ avec le nouveau vecteur de coûts réduits c'_j

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
c'_j	1	1	1	0	1	0	1	0	0	0	0	2	1	0	1	0

j	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
c'_j	3	0	0	1	1	4	0	1	0	1	5	1	5	2	0	0

Optimalité

- cas du partitionnement : pour trouver une solution meilleure que celle déjà obtenue, il ne faut utiliser que des arcs de coûts nuls ($c'_j = 0$)

d'où $f(6) = 6$ OU $f(6) = 32$ (ligne 6)

et $f(7) = 25$ OU $f(7) = 31$ (ligne 7)

$f(6)=32$ $f(7) \neq 31$ et $f(7) \neq 25$ impossible donc

$f(6)=6$ si $f(7) = 31$ impossibilité donc

$f(7)=25$ qui mène de nouveau à une impossibilité (ligne 3)

la solution

$$x_{23} = x_{25} = x_{26} = 1 \qquad \text{est donc optimale.}$$

• cas de recouvrement : seuls les coûts de $\underline{5}$ sont éliminés mais le programme obtient tout de suite une solution meilleure par les choix :

$$f(14)=19 \ , \ f(10)=25 \ , \ f(15)=18 \ , \ f(6)=6 \ , \ f(9)=8 \ , \ f(5)=5$$

cette solution vaut 14. Comme dans le cas précédent les arcs menant à des colonnes de coûts non nuls sont alors supprimés et le programme démontre alors l'optimalité de cette dernière solution.

Il y a dans le problème de recouvrement moins d'implications après chaque choix, mais la preuve de l'optimalité est plus longue que dans le premier cas et au total les temps de calcul sont presque identiques : 4 secondes.

Des données numériques concrètes provenant de la RATP [Heurgon (1972)] ont par ailleurs été testées pour le problème de partitionnement.

	n	m	temps en secondes	nb. de remontées
Pb1	327	32	47	128
Pb2	327	43	86	628
Pb3	512	45	91	815
Pb4	878	130	186	2464

Autres fonctions économiques

La fonction "somme des coûts" n'est pas toujours utilisée dans ces problèmes; on lui préfère souvent la fonction dit "bottleneck"

$$\min Z \qquad Z = \max_{j=1,\dots,n} c_j \, x_j$$

On a alors, après réduction, simplement

$$\underline{Z} = \text{maximum des quantités soustraites,}$$

et dans le problème de recouvrement bottleneck, le programme prouve ainsi directement l'optimalité de la solution obtenue par les seuls sommets de coûts réduits nuls.

Un problème d'allocation de ressources :

On dispose de trois batteries de disques de quatre emplacements de disques chacune.

Comment disposer n disques ($n \leq 12$) sur ces batteries en respectant les contraintes d'alimentation des canaux :
le taux d'alimentation alimentant chaque batterie est pris égal à la somme des taux moyens d'occupation des disques affectés à cette batterie.

batterie x

A_x = taux d'occupation maximum admissible du canal x en régime batch.

B_x = id° en régime temps réel.

batterie y

a_i = taux d'occupation moyen observé du disque i en batch.

b_i = id° en temps réel.

batterie z

La modélisation classique de ce problème consiste à prendre pour inconnues x_i , y_i , $z_i \in \{0,1\}$ pour $i \in [1,n]$: $x_i = 1 \Leftrightarrow$ disque i sur la batterie x et donc les contraintes (1), dites de "choix multiples" :

$$(1) \qquad x_i + y_i + z_i = 1 \qquad \forall\, i \in [1,n]$$

Les contraintes d'alimentation s'écrivent :

$$(2) \qquad \begin{array}{l} a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq A_x \\ a_1y_1 + a_2y_2 + \ldots + a_ny_n \leq A_y \\ \vdots \qquad\qquad\qquad\qquad\qquad \vdots \\ b_1z_1 + b_2z_2 + \ldots + b_nz_n \leq B_z \end{array}$$

On impose enfin :

$$(3) \qquad \sum_{i\in[1,n]} x_i \leq 4 \quad , \quad \sum_{i\in[1,n]} y_i \leq 4 \quad , \quad \sum_{i\in[1,n]} z_i \leq 4 \quad .$$

Ce modèle comporte donc 3n inconnues et (9+n) contraintes.

On ne fait ainsi que transcrire la simple recherche d'une application f de l'ensemble D des disques sur l'ensemble E des batteries. Chaque point image étant atteint au plus quatre fois option DMA (degré maximum). L'instruction

(3) tro fon f → D E DMA

remplace donc les contraintes (1) et (2).

Il reste les contraintes d'alimentation qui s'écrivent (contraintes du type III) :

(4) $\forall j \in E \quad \sum_{\substack{i \in D \\ f(i)=j}} a_i \leq A_j$ et $\sum_{\substack{i \in D \\ f(i)=j}} b_i \leq B_j$

Il est possible de poser le problème au programme sous la forme classique (1)(2)(3) , mais la formulation (3)(4) est plus naturelle, plus élégante et finalement plus efficace; c'est cette dernière qui est préférée.

Exemple : n=6

	1	2	3	4	5	6
a_i	0,25	0,20	0,12	0,08	0,05	0
b_i	0,12	0,10	0,15	0	0,05	0,02

avec Ax = 0,30 Ay = 0,20 Az = 0,30
Bx = 0,15 By = 0,2à Bz = 0,15

Le programme obtient les quatre solutions en une seconde.

Par exemple :

(5)	
(2)	

(3)	
(4)	

(1)	
(6)	

Le programme obtient des solutions admissibles à un problème avec 10 batteries et 40 disques en 5 secondes.

IV.2.2. Problèmes de type choix d'investissements.

Il s'agit de déterminer parmi un ensemble de projets, qui entrainent des charges connues et rapportent en revanche un gain connu, le sous ensemble, respectant la charge limite, qui assure le profit optimal. Mathématiquement, on cherche

$$\max Z = \sum_j c_j x_j$$

avec $x_j = 0,1 \qquad \forall j = 1,\ldots,n$

et $\sum_{ij} d_{ij} x_j \leq L_i \qquad \forall i = 1,\ldots,m$ si il y a m ressources.

Ce problème est la généralisation du cas particulier bien connu et étudié, dans lequel m=1 appelé "Knapsack problem".

Senju et Toyoda [1968] , puis Toyoda [1975] ont proposé des problèmes de ce type et une heuristique dite de "gradient effectif" pour obtenir d'excellentes solutions approchées.

La méthode consiste à mettre à un la variable qui possède le plus grand gradient projeté sur OL dans $\mathbb{R}^m$, L étant le point de coordonnées $(L_1, L_2 \ldots L_m)$. Toyoda impose, en outre, à chaque itération un changement de l'origine des coordonnées pour éviter une erreur systématique. Nous avons repris les mêmes données numériques (m=30 , n=60) . Le programme trouve d'abord une solution comportant 20 variables à 1 de valeur économique 7727, qu'il améliore ensuite, la solution passant à 7741 pour 21 variables (solution approchée identique à celle de Toyoda [1975] et préjugée optimale), le temps de calcul total est de 46 s. , le programme est arrêté sans rechercher l'optimum ce qui demanderait un temps trop long étant donnée la taille du problème. (une solution de valeur 8710 est obtenue en 49 s. dans le cas de la variable B pour le même problème, cf. données).

D'autres essais ont été effectués sur des problèmes plus petits, dont la solution optimale était connue et a été retrouvée le plus souvent par une descente directe. L'heuristique "difficultés-intérêts" utilisée, décrite en III.2, qui se simplifie d'ailleurs dans le cas linéaire, est donc efficace et assez générale pour être utilisée dans d'autres types de programmes mathématiques (cf. IV.1.5).

	i=1														i=15
	47	73	36	47	61	698	71	33	16	45	11	410	553	423	967
j=1	774	67	42	114	720	53	37	27	36	51	51	89	63	6	654
	76	27	56	26	7	290	978	13	538	59	97	414	35	58	46
	56	99	96	668	31	3	93	57	210	21	26	81	63	36	69
	59	35	438	82	22	62	9	6	844	53	83	130	58	37	26
	22	794	49	43	82	37	23	87	520	43	426	451	402	321	769
	42	53	57	55	688	704	47	21	33	25	92	6	664	22	82
	1	378	16	55	19	810	71	12	73	7	39	238	85	26	89
	21	234	978	56	82	42	744	15	100	42	66	79	458	16	15
	760	32	9	27	654	17	9	290	284	27	2	43	834	27	87
	818	792	644	716	809	983	619	206	650	310	23	5	3	218	46
	62	97	584	7	99	21	32	420	85	45	93	288	62	530	59
	42	64	82	77	81	45	214	32	894	72	85	910	508	93	22
	36	19	550	26	97	56	31	880	2	53	660	204	7	55	840
	785	435	283	950	830	234	796	854	206	798	85	948	1	85	66
	29	712	340	320	826	389	103	417	637	304	774	19	72	71	35
	662	837	596	350	775	712	593	770	324	354	77	644	88	828	684
	49	22	788	95	72	664	16	4	318	79	77	21	45	75	57
	608	504	33	714	9	59	468	12	7	45	927	295	496	628	254
	116	332	350	789	719	15	700	952	566	44	868	11	43	67	230
	834	13	55	430	740	22	884	604	46	52	7	6	750	66	21
	57	65	59	97	860	91	67	13	818	76	554	595	222	622	586
	42	86	348	590	72	57	36	96	92	45	760	904	96	440	51
	39	29	66	32	30	784	3	910	65	26	104	67	31	91	19
	594	894	468	69	82	75	93	34	520	27	48	51	278	73	984
	690	266	983	264	112	719	71	460	721	968	202	703	184	790	156
	27	75	6	19	68	294	734	76	90	86	45	420	36	710	23
	524	16	33	51	638	978	504	16	53	16	75	95	7	59	748
	23	86	42	97	150	75	81	140	174	62	51	408	210	83	760
	96	91	96	33	13	86	53	100	43	85	55	89	55	968	65
	667	67	464	571	586	165	509	876	320	790	716	11	653	129	339
	490	145	175	388	590	227	114	622	812	208	752	495	51	632	892
	805	118	33	602	519	760	293	559	382	390	37	844	35	170	13
	40	72	97	140	2	2	31	39	16	36	95	13	37	67	13
	19	97	16	13	320	120	73	340	378	92	237	417	292	612	327
	26	370	22	85	13	3	59	59	678	83	91	570	2	608	65
	97	88	9	42	964	71	99	6	29	93	5	9	49	43	35
	71	85	554	66	754	32	11	244	92	16	956	429	884	71	246
	699	165	358	778	70	210	67	232	755	574	444	16	418	730	71
	465	268	587	936	241	678	306	513	827	150	529	939	379	910	178
	53	758	71	61	72	41	96	644	27	99	96	430	75	36	83
	26	21	23	23	12	93	218	7	218	7	99	270	338	92	3
	123	255	931	449	996	47	845	313	143	920	17	49	51	950	34
	20	81	931	973	868	581	303	624	12	860	99	72	21	138	624
	25	5	94	828	36	37	3	990	57	12	62	65	29	23	788
	450	774	798	33	91	977	319	274	480	404	7	66	95	95	200
	22	39	73	53	79	62	86	808	154	31	394	338	790	460	980
	979	377	873	297	221	503	724	372	944	560	580	994	846	62	882
	75	18	22	75	49	32	22	2	7	37	604	167	720	189	343
	96	370	39	3	690	85	838	694	730	32	89	76	71	73	550
	27	96	71	54	23	31	82	804	12	9	678	47	728	65	708
	41	61	864	27	18	36	5	39	65	62	476	211	930	943	542
	21	57	59	918	748	30	330	5	67	7	97	87	95	62	53
	81	23	82	11	408	328	58	644	39	43	234	39	1	554	72
	15	13	16	620	688	74	55	914	390	17	1	1	910	46	86
	76	164	444	13	97	31	66	484	22	77	608	570	5	318	51
	97	908	37	28	85	56	53	1	39	73	19	85	804	13	700
	646	834	475	80	777	891	916	8	318	368	69	134	5	540	524
	808	960	65	79	294	62	89	43	47	68	676	967	284	90	577
j=60	37	87	6	3	17	97	56	92	86	82	51	12	128	53	948

Matrice des d(i,j)

i=16															i=30
j=1	132	869	494	267	624	33	19	47	53	9	37	65	468	23	35
	900	26	2	290	7	62	780	75	12	19	36	59	13	93	55
	72	710	25	31	96	928	400	55	79	7	77	2	9	87	21
	51	37	61	844	45	292	588	554	481	535	843	671	120	490	264
	91	81	674	12	97	2	37	3	218	76	828	26	73	32	5
	150	89	638	570	148	340	86	800	26	3	22	364	7	63	35
	22	6	61	13	91	278	23	6	624	27	76	854	92	870	92
	110	82	59	69	3	911	583	13	954	928	368	526	99	393	128
	154	82	62	65	69	818	818	85	13	21	39	570	93	52	65
	148	56	690	848	26	770	42	65	580	7	71	630	418	23	27
	99	96	630	72	22	464	56	99	130	864	35	33	224	63	9
	75	66	86	780	45	53	1	45	608	27	850	654	22	634	52
	21	46	198	27	12	888	108	69	3	73	96	95	7	39	66
	544	13	24	96	2	65	83	73	37	61	93	41	29	83	51
	110	934	15	97	75	76	43	864	91	25	87	42	57	12	7
	11	49	650	17	76	31	720	95	78	75	56	27	33	72	47
	52	394	75	69	96	389	570	199	743	876	972	584	949	131	670
	840	72	25	274	67	40	81	924	1	16	96	17	65	69	83
	201	194	571	616	688	864	674	19	950	92	594	724	92	984	76
	2	408	338	36	2	36	25	948	45	22	864	59	858	87	7
	6	5	268	554	2	35	96	214	41	248	344	42	200	86	79
	663	541	958	236	224	37	218	3	718	11	76	26	57	99	37
	22	88	95	47	83	69	6	718	36	86	17	918	12	52	2
	20	93	898	7	69	95	69	56	30	944	17	6	31	110	46
	89	36	56	47	41	22	107	278	534	872	576	242	296	183	480
	10	398	585	134	660	648	534	1	418	956	629	356	185	704	608
	93	508	99	76	81	334	158	363	452	252	780	75	272	232	990
	964	89	83	62	89	14	56	86	359	2	640	644	91	674	53
	924	66	21	824	93	198	5	1	759	800	56	818	77	384	47
	73	16	600	55	27	42	938	22	88	334	65	168	37	47	19
	501	71	462	374	214	73	9	56	29	93	43	964	734	804	35
	398	466	940	471	458	594	938	114	499	107	196	12	911	99	518
	3	7	96	478	66	95	274	13	55	254	520	97	27	83	71
	2	95	464	504	72	32	76	53	974	441	86	178	310	81	59
	279	484	228	496	284	814	298	56	92	930	92	634	59	174	32
	5	41	93	754	59	45	9	19	56	744	31	21	33	99	87
	288	69	72	604	76	45	518	82	108	97	6	3	87	77	96
	80	130	734	923	538	515	571	88	257	177	593	586	872	708	240
	91	488	89	330	15	634	47	99	93	498	174	47	73	7	52
	132	695	287	22	840	254	175	543	171	931	569	382	79	623	310
	620	82	174	97	65	42	63	674	13	694	89	804	920	114	86
	628	39	62	6	63	29	93	704	92	800	718	89	85	1	73
	57	95	51	2	77	15	49	418	63	9	83	194	59	750	52
	79	53	73	16	33	83	94	670	714	36	8	21	72	49	31
	2	37	42	14	92	55	42	554	9	6	790	610	888	284	83
	874	200	838	958	32	176	26	282	84	539	285	168	49	492	544
	36	87	82	53	35	35	79	5	890	35	780	79	12	11	16
	497	56	515	480	832	46	50	67	16	79	62	96	79	61	15
	846	268	232	482	970	60	718	63	930	130	378	87	538	6	21
	22	71	91	93	49	296	926	466	967	860	313	266	947	449	774
	350	304	25	57	13	262	419	491	748	710	519	482	462	429	224
	866	855	47	641	8	598	810	49	5	7	2	46	444	52	7
	57	22	12	72	77	67	23	67	7	630	85	96	828	62	83
	86	564	56	75	75	644	363	154	6	475	845	969	935	482	680
	83	47	65	51	51	80	519	956	327	903	931	629	518	826	554
	178	26	734	96	95	999	339	911	894	552	731	128	894	147	310
	968	26	70	83	56	3	86	77	33	2	42	924	13	338	96
	52	370	48	47	63	727	751	635	629	45	365	812	591	911	844
	399	569	209	403	578	79	7	2	448	97	32	19	22	30	29
	628	2	71	32	47	374	86	49	21	974	83	2	920	984	61

2 77 6 67 930 3 6 270 33 13 110 21 56 974 47
734 238 75 200 51 47 63 7 6 468 72 95 82 91 83
27 13 6 76 55 72 300 6 65 39 63 61 52 85 29
640 558 53 47 25 3 6 568 6 2 780 69 31 774 22

Coefficients c_j $j=1,\ldots,60$

6000 6000 6000 6000 6000 6000 6000 6000 6000 4000
6000 6000 6000 6000 6000 6000 6000 6000 6000 4000
6000 6000 6000 6000 6000 6000 6000 6000 6000 4000

Variante A : Coefficients L_j $j=1,\ldots,30$
Variante B : 6000 changés en 10000 , 4000 en 7000.

IV.2.3. Autres problèmes linéaires.

D'autres problèmes classiques de programmation en nombres entiers ont été soumis au programme. La manipulation formelle en polonaise est désavantageuse par rapport à la notation matricielle dès que le problème est linéaire. Les petits problèmes, tels que ceux donnés par Balas [1965] ou Hammer et Rudeanu [1968] sont résolus en deux ou trois secondes. L'optimalité ne peut pas, par contre être prouvée en 2 minutes, pour,par exemple,les problèmes de Haldi [cf Garfinkel et Nemhauser (1972)] , bien que la solution optimale ait été trouvée par l'heuristique "difficultés-intérêts". De même les résultats concernant le problème du knapsack ne peuvent rivaliser avec les algorithmes spécifiques utilisant des théorèmes fondés sur la solution continue.

Il existe pour les problèmes suivants en particulier, de bons algorithmes,et il n'est pas question de comparer ces algorithmes avec le programme, mais simplement de montrer qu'ils peuvent bien être posés :

problème du plus court chemin : ce problème est équivalent au problème central de l'ordonnancement.

problème de flot maximum .

problème de transport (ou d'affectation déjà cité) : Les algorithmes de stepping stone et de Ford Fulkerson sont très performants, mais l'heuristique (III.7) et les réductions et regrets conduisent à des solutions approchées qui peuvent être utilisées pour initialiser ces algorithmes.

problème de l'arbre minimum : les réductions et les simplifications redonnent la même solution que l'algorithme de Kruskal.

problème de couplage maximum : dans la plupart des cas testés, l'heuristique consistant à choisir une arête issue d'un des sommets de degré minimum donne une solution de cardinal $\left\lceil \frac{\text{nombre de sommets}}{2} \right\rceil$, et donc optimale.

Il est en revanche possible de soumettre grâce au langage ALICE, sous une forme autre que celle d'un système d'équations de nombreux problèmes. La modélisation en système d'équations est d'ailleurs une méthode détournée et peu satisfaisante, utilisée surtout dans le but d'économiser l'écriture de programmes spécifiques. Les paragraphes IV.3 à IV.9 présentent des énoncés sous une forme plus naturelle.

V.2.4. Programmation quadratique en nombres entiers.

La programmation non linéaire en entiers ne permet pas jusqu'ici de résoudre des problèmes de taille importante : une cinquantaine de variables booléennes semble être un maximum dans le cas général.

Balas (1964), Zangwill (1965), Glover et Woolsey (1972), Taha (1972) proposent de remplacer tout terme non linéaire par adjonction d'une nouvelle variable et de deux nouvelles contraintes dans le cas booléen.

Hammer et Rubin (1970) et Abadie (1971) suggèrent de se ramener à la programmation convexe, que Kelley (1960) et Gondran (1972) résolvent par l'intermédiaire de plans de coupes utilisant les équations de congruence.

Dakin (1965), Lawler et Bell (1966), Mao et Wallingford (1968), Hansen (1974) utilisent des méthodes purement énumératives.

L'heuristique générale pour trouver une solution réalisable entière à un système d'inéquations s'est révélée assez bonne sur ces problèmes. Le tableau de la page suivante montre les temps de calcul du programme et de ceux des

algorithmes de Mao et Wallingford d'une part et d'Hansen d'autre part.

Le fait que les temps de calcul puissent être favorablement comparés est dû à trois causes :

- la solution optimale est obtenue très rapidement par le programme et l'effort majeur est la démonstration de l'optimalité par preuve d'impossibilité d'améliorer encore la fonction économique. Tout au long de l'exécution, l'heuristique conduit en moyenne à de bons choix.
- l'ensemble des contraintes évolue constamment : le programme engendre de nouvelles inéquations, fait disparaître celles qui sont toujours satisfaites, élimine certaines variables.
- dès qu'il n'y a plus qu'un petit nombre de variables libres (ici environ 6 pour tous les problèmes) le programme termine par une énumération explicite.

N° du Pb	nombre de variables	valeur de Z à l'optimum	temps en secondes CDC 6500 Mao et Wallingford	temps en secondes CDC 6500 Hansen	temps en secondes IBM 370/168
1	6	10	0,14	0,11	4
2	8	27	0,85	0,35	12
3	10	28	2,04	0,34	11
4	12	21	9,6	1,16	24
5	14	12	30,7	0,56	26
6	16	5	103,2	0,52	20
7	18	17	483,2	1,9	29
8	20	7	>1000	1,3	35
9	22	2		0,7	9
10	24	7		6,8	39
11	26	9		8,3	52
12	28	17		42,7	137
13	30	10		9,7	30
14	32	31		110,6	92
15	34	12		115,6	105
16	36	12		162,1	104
17	38	14		401,6	235
18	40	15		729,2	309

La méthode lexicographique de Mao et Wellingford n'utilise aucune heuristique. Hansen cherche seulement une solution de départ en partant de $X \equiv 1$ et en fixant à zéro les inconnues qui diminuent le plus Z, tout en conservant l'admissibilité; les données sont construites de telle sorte que $X \equiv 1$ est solution admissible pour tous ces problèmes. Hansen travaille ensuite avec des pénalités et un système complexe de tests.

Données des 18 problèmes tests de programmation quadratique.

```
HANS
****
SET CST A
    ENS E   =   INT 1   A
        B   =   INT 0   1
    CCE CLF SUR E                ┐ coefficients des termes linéaires
        IQF SUR E                │ 1er indice variable des termes quadratiques
        JQF SUR E                │ 2ème  "      "        "      "      (fonction économi.)
        CQF SUR E                ┘ coefficients des termes quadratiques
        CL1 SUR E                ┐
        IQ1 SUR E                │
        JQ1 SUR E                │                                    contrainte 1
        CQ1 SUR E                │
        CL2 SUR E                ┘
        IQ2 SUR E                ┐
        JQ2 SUR E                │                                    contrainte 2
        CQ2 SUR E                │
        CL3 SUR E                ┘
        IQ3 SUR E                ┐
        JQ3 SUR E                │                                    contrainte 3
        CQ3 SUR E                │
        CL4 SUR E                ┘
        IQ4 SUR E                ┐
        JQ4 SUR E                │                                    contrainte 4
        CQ4 SUR E                ┘
TRC FCN X   CEA E   B
AVE MIN SIG APP I   E   +   *   CLF I   X   I   *   CQF I   *   X   IQF I   X
JQF I
AVE <=  SIG APP I   E   +   *   CL1 I   X   I   *   CQ1 I   *   X   IQ1 I   X
JQ1 I   5
AVE <=  SIG APP I   E   +   *   CL2 I   X   I   *   CQ2 I   *   X   IQ2 I   X
JQ2 I   45
AVE <=  SIG APP I   E   +   *   CL3 I   X   I   *   CQ3 I   *   X   IQ3 I   X
JQ3 I   45
AVE <=  SIG APP I   E   +   *   CL4 I   X   I   *   CQ4 I   *   X   IQ4 I   X
JQ4 I   40
FIN
34,
2,1,8,7,6,8,5,9,4,5,8,2,1,8,5,3,5,1,1,6,9,2,6,9,7,8,2,7,8,3,2,8,5,2,2,9,4,4,5,3,
2,4,2,5,1,3,6,2,4,1,8,6,11,9,3,8,14,3,9,17,2C,6,2,8,21,4,15,4,17,26,
8,22,3,3C,17,13,9,27,33,25,
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,2C,21,22,23,24,25,26,
27,28,29,3C,31,32,33,34,35,36,37,38,39,4C,
3,-1,8,-4,8,-7,7,-3,4,-3,7,-6,3,-8,9,-6,1,-3,9,-4,6,-7,4,-6,6,-7,2,-6,5,-5,4,-2,
2,-5,4,-3,4,-5,7,-3,
3,-9,7,-5,-8,3,9,5,7,-9,-7,6,9,-9,-6,-9,2,-3,7,8,9,-6,4,-4,-3,2,-7,-7,-7,1,
9,1,-8,2,-3,3,-9,5,-9,7,
3,4,5,1,6,2,3,5,8,4,9,2,3,5,1C,11,4,5,13,14,20,15,7,4,16,7,8,21,4,19,
1C,3,5,17,15,16,21,5,23,32,
1,2,3,4,5,6,7,8,9,10,11,12,13,15,15,16,17,18,19,2C,21,22,23,24,25,26,27,28,29,3C
31,32,33,34,35,36,37,38,39,4C,
8,-8,5,-9,8,-7,4,-5,8,-2,6,-6,2,-8,1,-2,5,-8,9,-3,9,-4,8,-4,8,-7,4,-3,4,-2,
9,-1,8,-2,3,-9,7,-9,5,-4,
```

```
-6,4,7,4,6,8,9,-7,9,6,6,-1,9,7,6,6,-1,4,2,6,-8,-9,-7,7,4,9,
4,1,-2,-8,4,-9,-4,7,-8,4,
5,3,6,2,3,2,5,6,7,2,3,7,8,5,9,8,4,5,15,3,10,11,17,6,4,9,13,24,7,4,
8,9,13,22,28,34,19,24,17,8,
1,2,3,4,5,6,7,8,9,10,11,12,13,15,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30
31,32,33,34,35,36,37,38,39,40,
9,-4,1,-6,5,-3,9,-7,1,-4,1,-2,6,-8,7,-7,5,-6,3,-7,3,-3,4,-8,8,-5,7,-5,1,-5,
4,-4,1,-1,6,-9,6,-4,7,-8,
-3,4,5,1,-9,8,4,-3,5,-1,7,3,7,5,5,-4,1,7,1,-9,-7,6,5,-6,4,-8,-7,2,7,7,
6,2,7,5,4,7,3,1,2,4,
4,5,4,2,1,3,3,6,4,9,4,7,3,2,10,4,5,3,10,8,7,6,15,16,4,14,21,25,21,3,
18,15,18,23,9,19,4,29,31,35,
1,2,3,4,5,6,7,8,9,10,11,12,13,15,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30
31,32,33,34,35,36,37,38,39,40,
5,-7,1,-7,6,-8,4,-2,6,-1,3,-4,9,-9,2,-7,9,-3,6,-5,4,-6,4,-9,1,-5,9,-6,7,-7,
5,-9,6,-1,8,-4,4,-9,6,-6,
4,4,-3,-6,5,5,-5,6,9,3,3,7,8,4,3,9,8,6,6,-7,9,-6,-9,6,7,-8,-8,-4,9,5,
6,2,-9,-2,1,-6,-7,-8,6,6,
2,5,4,5,3,5,2,3,4,7,4,3,4,10,2,1,8,9,15,2,7,12,11,5,22,4,13,13,12,26,
4,29,1,15,33,17,5,22,35,5,
1,2,3,4,5,6,7,8,9,10,11,12,13,15,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30
31,32,33,34,35,36,37,38,39,40,
9,-8,5,-5,4,-2,6,-9,3,-3,4,-7,1,-4,4,-1,8,-4,2,-1,2,-5,1,-5,4,-8,3,-1,1,-2,
7,-8,2,-9,1,-3,2,-1,4,-8,
FIN
```

Le problème n°i possède n=4+2*i variables.

Il y a à chaque fois 4 contraintes.

Contraintes et fonction économique comportent chacune n termes linéaires, n/2 termes quadratiques à coefficients positifs, n/2 termes quadratiques à coefficients négatifs. Quand on passe du problème i au problème i+1 , on conserve les anciens coefficients et on y ajoute donc 2 valeurs par vecteur (CLF à CQ4 cf. énoncé). Les valeurs des seconds membres b_j des contraintes (j=1 à 4) évoluent, sont donnés dans le tableau :

Pb	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
b_1	5	10	20	-5	30	0	45	25	40	0	35	45	50	85	5	20	10	15
b_2	5	5	15	30	15	10	25	20	5	15	25	20	30	30	45	15	55	15
b_3	5	5	25	-5	25	5	15	25	5	-5	20	10	20	20	45	15	40	15
b_4	5	15	0	5	0	15	15	10	5	35	25	35	30	25	40	50	30	60

IV.3. PROBLEME D'ORDONNANCEMENT AVEC DATE LIMITE, DE DECOUPE LINEAIRE , "BIN PACKING"

Pb 1 : On veut organiser sur un ordinateur multiprocesseur le passage d'un ensemble de tâches de durées connues en respectant une date limite donnée.

Pb 2 : Disposant de grandes barres de longueur connue, on cherche à découper dans un nombre minimum de ces grandes barres des morceaux plus petits en quantités et longueurs données.

Exemples

Pb 1 : 9 tâches de durées 13 , 15 , 9 , 6 , 6 , 8 , 6 , 3 , 2 doivent être exécutées avant la date limite (deadline) 18 .

```
DECCLPE
*******
SCT CST N    L
    ENS T    =    INT 1    N
    CCE C    SUR T
TRC FCN F    >-> T    T
AVE MIN MAX F    I
AVC <=  SIG ET  APP I    T    =    F    I    J    C    I    L        (1)
FIN
9,18,
13,15,9,6,6,8,6,3,2,
```

La contrainte (1)

$$\sum_{\substack{i \in T \\ fi=j}} d(i) \leq L$$

est du type II.3 , type prévu une fois pour toutes dans le programme.

En sommant sur tous les j , le programme obtient d'abord :

$$\sum_{i \in T} d(i) \leq \underline{Z} * L$$

où $\underline{Z}$ est une évaluation par défaut de l'optimum cherché soit ici

$$\underline{Z} \geq \left\lceil \frac{68}{18} \right\rceil \Rightarrow \underline{Z} = 4$$

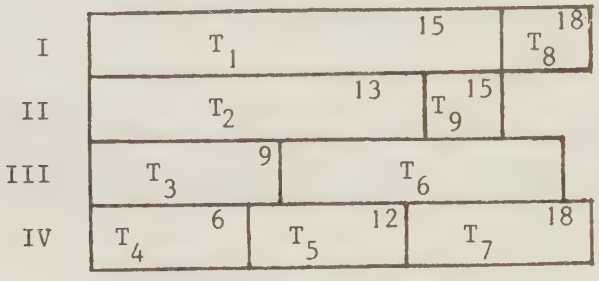
(Il faut au minimum quatre processeurs pour respecter la date de fin 18).

Dans φ_1 le programme trouve immédiatement la solution sans parallèlisme : fi = i $\forall$ i (9 processeurs).

Dans $\varphi 1$, le programme retient les critères :

pour i max durée
pour j max $|\{i|fi=j\}|$;

ce dernier critère directement déduit de (1) tend à utiliser au maximum le même processeur tant que cela est possible, ce qui conduit à :



qui est optimale.

Pb 2 : Dans des grandes barres de longueur 396 on désire découper des morceaux plus petits de 11 longueurs différentes :

type	1	2	3	4	5	6	7	8	9	10	11
longueur	285	188	126	115	112	75	60	51	12	10	9
nombre désiré	1	6	18	3	3	1	1	1	3	6	12

Il vient $\underline{Z} = 12$ et la solution optimale est obtenue directement dans ($\varphi 2$) par les mêmes critères (heuristique dite "first fit decreasing" dans Coffman [1975]).

Mais ce qui est intéressant dans ce cas pathologique construit par Sylvia Halasz [1975] , c'est que si l'on enlève le type 6 de longueur 75 en diminuant donc la demande, le nombre de barres donné par la même heuristique augmente.

On avait en effet :

1 fois	285 + (75)	+ 12 + 12 + 12
3 fois	188 + 188	+ 10 + 10
6 fois	126 + 126 + 126	+ 9 + 9
1 fois	115 + 115 + 115	+ 51
1 fois	112 + 112 + 112	+ 60

et tout s'était passé au mieux puisqu'on avait

$$\Sigma \text{ longueur} = 396*12 \quad \text{exactement.}$$

L'absence du morceau de 75 conduit à un décalage à droite du trait qui mène finalement dans $\varphi 2$ à une découpe utilisant cette fois 13 barres (la dernière sert uniquement pour un morceau de 9).

Ce phénomène provoque des calculs longs et difficiles sur l'écart maximum par rapport à l'optimum du"first fit decreasing"(ou du "best fit decreasing"), à propos de complexité des algorithmes dans la littérature [Johnson (1972), Lieberherr (1971)].

Le programme retrouve une solution en 12 barres dans la phase $\varphi 3$ presque immédiatement.

Cet exemple met bien en évidence que, dans la résolution d'un problème combinatoire, une heuristique seule ne suffit pas (par définition) et qu'une arborescence est indispensable: cela ne coûte d'ailleurs pas cher tant que les choix ne sont pas trop mauvais et que le nombre de retours arrière n'est pas trop important.

Genuys [1972] propose un exemple avec 647 morceaux à découper dans des barres de longueur 800 :

type	1	2	3	4	5	6	7	8
de longueur	224	124	114	90	83	82	70	44
en nombre	11	17	15	46	51	46	270	191

Il propose de résoudre ces problèmes par une méthode duale du simplexe puis des congruences.

Le modèle linéaire demande 15 itérations principales (appels du simplexe) et 23000 itérations auxiliaires (pivotages par congruences) environ.

Le programme trouve ici $\underline{Z} = 58$, ce qui autorise une chute de 669 unités, et obtient une solution en 60 barres dans $\varphi 1$ en 59 dans $\varphi 2$. La solution optimale en 58 est obtenue finalement dans $\varphi 3$ après 951 retours en arrière pour un temps total d'exécution de 41 secondes.

Il est clair que si les données changent peu, l'exécution n'est pas fortement modifiée, tout au plus le nombre de remontées dans $\varphi 3$ augmente-t-il. Il

n'en va pas de même dans le modèle de programmation linéaire, où la solution continue peut alors évidemment varier beaucoup, ainsi que le temps nécessaire aux congruences.

IV.4 - PROBLEMES D'ORDONNANCEMENT DISJONCTIFS.

Les seules contraintes dans le problème "central" sont de la forme

$$f(i) \geq f(j) + \text{durée } (j)$$

où $f(i)$ est la date au plutôt de la tâche i , une telle contrainte exprimant que i est postérieure à j . Le traitement des contraintes d'effectue, contrairement à la méthode PERT , par l'intermédiaire de la fonction de Grundy et le programme retrouve donc la méthode dite "des potentiels" et l'optimalité de

$$\min \max_{i \in D} f(i)$$

est prouvée ainsi directement.

Dans les problèmes disjonctifs interviennent les contraintes :

$$f(i) \geqslant f(j) + \text{durée } (j) \quad \underline{\text{OU}} \quad f(j) \geqslant f(i) + \text{durée } (i)$$

qui expriment la disjonction des tâches i et j .

La recherche arborescente s'effectue en deux temps :

- arbitrage des contraintes OU
- résolution du problème central associé.

Mais, dans l'état actuel du programme, il n'est possible de prendre en compte avec un temps de calcul raisonnable que quelques contraintes du dernier type.

Une modification, qu'il serait logique d'apporter au programme, permettrait de mémoriser ce type de contraintes dans le graphe interne, par une généralisation des cliques de disjonctions :

Il conviendrait de valuer tout arc de disjonction entre les sommets i et j par durée (i) et durée (j) , ce qui est simple à réaliser, mais il faut ensuite faire intervenir , et c'est là un point plus délicat, ces valuations dans le calcul du minorant $\underline{Z}$ après chaque arbitrage. Cette façon de faire a été étudiée par J.Carlier dans sa thèse de 3° Cycle (1975).

IV.5 - PROBLEMES DE DECOUPE ET AIDE A LA CONCEPTION.

Charles E. Pfefferkorn [1975] a écrit un programme, le "designer problem solver" (DPS) , pour disposer des éléments dans un espace en respectant un ensemble de conditions : aménagement de locaux d'habitation, dessin d'immeubles, et plus simplement problèmes de découpe dans un plan.

DPS dans une première phase définit une stratégie, rep-ère les difficultés et réduit la taille du problème par fabrication de macro-objets, il utilise des heuristiques pour réduire la taille de l'arborescence.

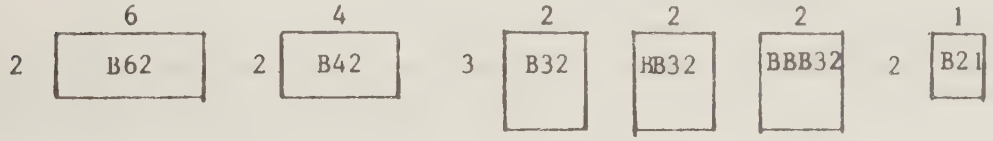
Pfefferkorn donne essentiellement deux types de problèmes différents, que nous avons traduit en ALICE .

Cas 1 - Découpe :

Ce premier type de problème consistant à découper dans une plaque rectangulaire (D1 sur D2) des morceaux plus petits de dimensions connues, se formule en fait très facilement en programmation linéaire en nombres entiers. L'approche par une méthode de congruences de Gilmore et Gomory [1963] et [1965] fondée sur le simplexe donne lieu à un programme assurément plus performant dans l'état actuel des programmes de Pfefferkorn et du mien. Mais la prise en compte, dans ce modèle, de contraintes additionnelles (cas 2) est très délicate au point de ne pas être abordée en recherche opérationnelle jusqu'ici.

Exemple

Placer les 6 blocs suivants dans une boîte de 5x8



Fikes proposait un problème semblable plus évident . Chaque bloc, défini par ses deux dimensions (vecteurs LON et LAR), est repéré dans la boîte par les coordonnées d son coin en bas à gauche (fonctions BAS et GAU) et par son orientation. Les chevauchements sont interdits en écrivant qu'un bloc est ou à gauche, ou à droite, ou en haut, ou bien enfin an bas d'un autre bloc (contrainte (2) , (1) n'est qu'une commodité d'écriture manuelle pour définir complètement la position d'un bloc).

```
ARCHIT
******
SCI CST N   C1  C2
    ENS EO  =   INT 1   N                       ensemble des blocs
        EC  =   INT 0   1
        E1  =   INT 1   -   N   1
        FO  =   INT 0   -   C1  1
        GO  =   INT 0   -   C2  1
        F1  =   INT 1   C1
        G1  =   INT 1   C2
    CCE LON SLR EC                              longueur
        LAR SLR EO                              largeur des blocs
TRC FCN GAU >-> EC  FO                          abcisse du point en bas à gauche
        BAS >-> EC  GO                          ordonnée
        DRC >-> EO  F1                          abcisse du point en haut à droite
        HAU >-> EO  G1                          ordonnée
        ORI >-> EC  BC                          orientation (verticale ou horizontale)
AVE CCS APP I   EO  ET  =   DRC I   +   GAU I   +   *   LON I   ORI I   *   LAR
I   -   1   ORI I   =   HAU I   +   BAS I   +   *   LAR I   ORI I   *   LON I
-   1   ORI I                                                           (1)
AVE CCS APP I   E1  CCS APP K   INT +   I   1   N   CU  <=  DRC I   GAU K   CU
<=  DRC K   GAU I   CU  <=  HAU I   BAS K   <=  HAU K   BAS I       (2)


FIN
6,8,5,
6,4,3,3,3,1,
2,2,2,2,2,2,
FIN
```

Les critères retenus par le programme sont

pour i : minimum de libertés
variable figurant dans le maximum de contraintes (pondérées par leur complexité)
plus grande dimension.

pour j : heuristique des coins (bascule).

La contrainte (1) donne pour tout i les règles de réécriture

dro i → gau i + lon i * ori i + lar i * (1 - ori i)

et hau i → bas i + lar i * ori i + lon i * (1 - ori i)

avec $1 \leq dro\ i \leq 8$ et $1 \leq hau\ i \leq 5$.

Ce qui permet d'écrire, par exemple pour i=5 :

$$\underbrace{bas\ 5}_{[P,PP]} - \underbrace{4*ori\ 5}_{4*[0,1]=[N,NN]} + 1 < 0 \quad \text{(Cf. II.2.2)}$$

donne naissance à :

bas 5 ≤ 3 et à :

- 4*ori 5 ≤ -1 démontrant ainsi que

ori 5 = 1

dans toute solution.

Le jeu des critères 1 et 2 provoque alors un choix sur la variable ayant peu de libertés et intervenant dans le plus de contraintes peu complexes, soit gau 5 , qui est fixée à la première valeur possible.

gau 5=0 (dro 5=6)

Puis, de même : bas 5=0 (hau 5=2)

La contrainte (2) , qui s'écrivait pour i=2 et k=5 :

dro 2 ≤ gau 5 ou dro 5 ≤ gau 2 ou hau 2 ≤ bas 5 ou hau 5 ≤ bas 2

devient par substitutions et simplifications :

6 ≤ gau 2 ou 2 ≤ bas 2 .

L'analyse des autres contraintes ne donnant rien, c'est cette dernière contrainte ou qui donne lieu à un choix :

6 ≤ gau 2

Alors par (1) pour i=2 :

$$(\text{dro } 2 =) \underbrace{\text{gau } 2}_{6,7} + \underbrace{2*\text{ori } 2}_{0,2} + 2 \leq 8$$

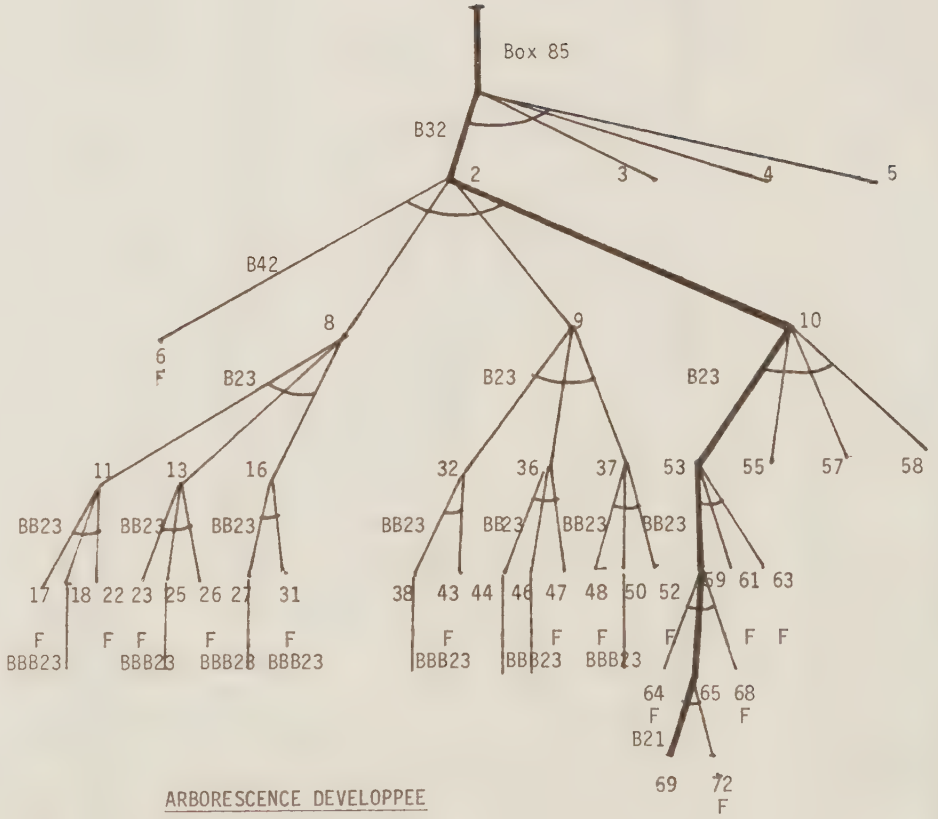
donc ori 2 = 0

et gau 2 = 6

La valeur de bas 2 est fixée à son maximum par l'heuristique des coins :

bas 2 = 2

et le programme obtient finalement , en fixant l'une après l'autre les positions des 3 blocs B32 et en déduisant celle de B12.



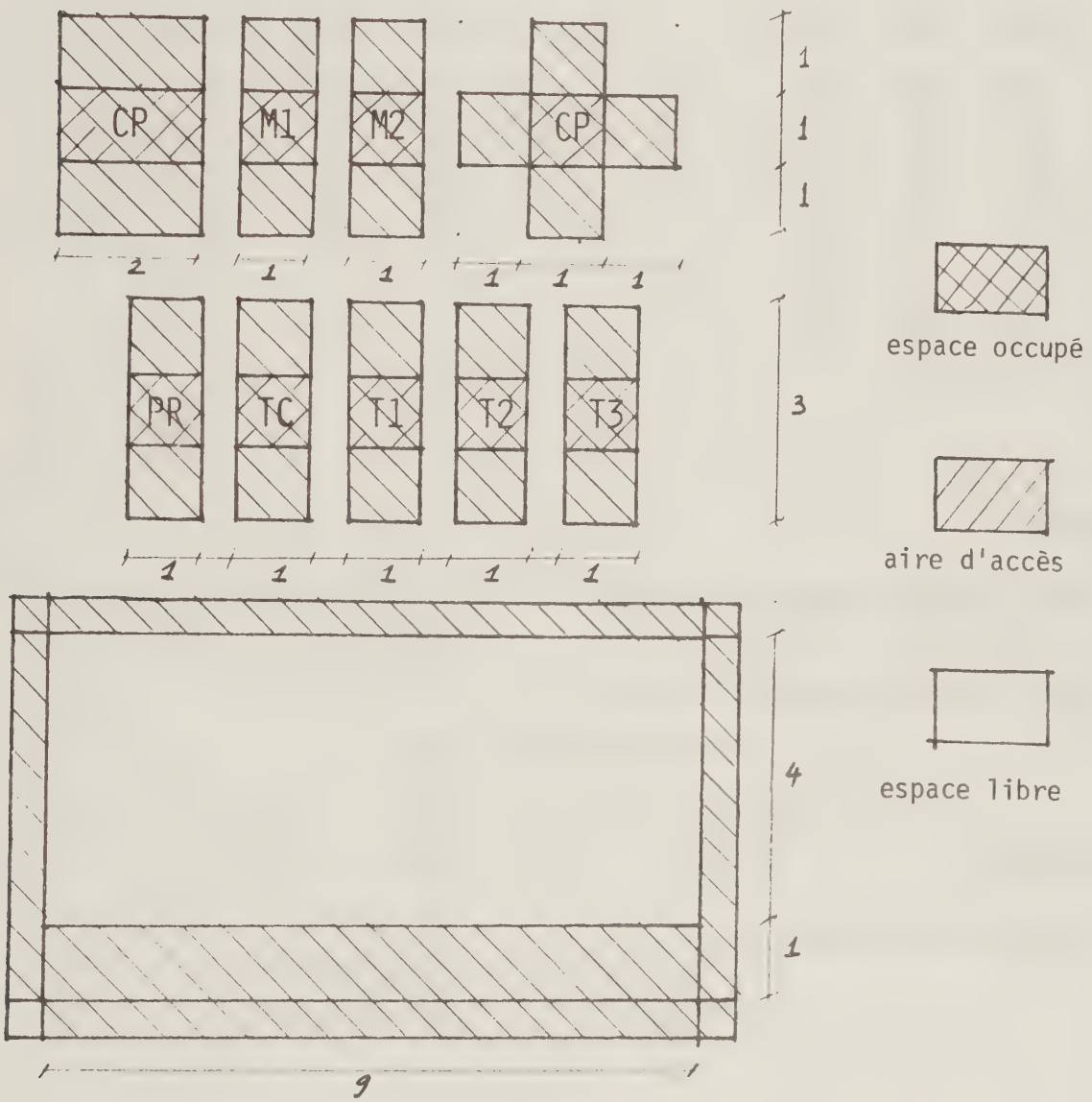
ARBORESCENCE DEVELOPPEE
PAR DPS POUR LE CAS 1

ori 5 = 1 (B62)
gau 5 = 0 (B62)
gau 2 ≥ 6 (B42)
ori 2 = 0
gau 2 = 6
bas 2 = 2 (B42)
gau 6 = 0 (BBB23)
gau 4 = 2 (B23)
gau 5 = 4 (BB23)
gau 1 = 6 (B21)

ARBORESCENCE DEVELOPPEE PAR NOTRE
PROGRAMME DANS LE MÊME CAS

s 2 - Agencement d'un plan au sol :

Neuf équipements (une unité centrale (CP) , un lecteur de cartes (CR) , une primante (PR) , deux modules mémoire (M1) et (M2) , quatre dérouleurs de ndes magnétiques (TC) (T1) (T2) et (T3)) doivent être placés dans une salle ordinateurs en respectant des conditions de position, d'accès et de distances tuelles :



Contraintes

Distances : CABLE - distance du câble de connexion entre OBJ1 et OBJ2 inférieur à Maxd.

OPER - distance entre les consoles de l'opérateur des OBJ1 et OBJ2 inférieure à Maxd.

CABLE

OBJ1	OBJ2	Maxd
CP	M1	2
MI	M2	2
TC	T1	2
T1	T2	2
T2	T3	2
CP	TC	10
CP	CR	5
CP	PR	10
pièce	CR	4

OPER

OBJ1	OBJ2	Maxd
CP	CR	5
CP	PR	10
T1	T2	5
T2	T3	5
T3	T1	5
CR	PR	3

Position

M1 dans la moitié gauche de la pièce
M2 " " " "
PR dans la moitié basse de la pièce
CR " " " "

Orientation

CP	en face de la console de	T1
CP		T2
CP		T3
CP		CR
CP		PR
T1	à l'horizontale de	T2
T1		T3
T1		TC
T2		T1
TC		T1
CR		PR
PR		CR
CR	devant	CP
M1	derrière	CP
M2	derrière	CP

```
SCI CST N   C1  C2  M
    ENS CC  =   INT 1   M
        MAT =   CEJ
        EO  =   INT 1   N
    CCE LCN SUR EC
        LAR SUR EC
        CB1 SUR CC
        CB2 SUR CC
        LIS SUR CC
        CRI SUR EO
    ENS E1  =   INT 1   -   N   1
        FO  =   INT 0   -   C1  1
        GO  =   INT 1   -   C2  1
        F1  =   INT 1   C1
        BC  =   INT 0   1
TRC FCN CAL >-> EO  FO
        BAS >-> EC  CO
        CRC >-> EC  F1
        HAL >-> EC  GO
AVE CCS APP I   EC  --> CRC I   +   GAL I   +   *   LCN I   CRI I   *   LAR I
-   1   CRI I
AVE CCS APP I   EC  --> HAU I   +   BAS I   +   *   LAR I   CRI I   *   LCN I
-   1   CRI I
AVE CCS APP I   E1  CCS APP K   INT +   I   1   N   CL  <=  CRC I   GAL K   CL
<=  CRC K   GAL I   CU  <   HAU I   BAS K   <   HAL K   BAS I
AVE <=  CRC 2   4
    ==> =   CRC I   J   CUI
    <=  CRC 3   4
    <=  HAL 4   /   C2  2
    <=  HAL 5   /   C2  2
    CCS APP I   INT 4   9   ET  <=  HAL 1   +   HAL I   1   >=  HAL 1   -   HAL
I   1
AVE CCS APP J   CC  CU  <=  GAU CB1 J   +   CIS J   GAL CB2 J   <=  GAL CE2 J
+   CIS J   GAL CE1 J
AVE ET  =   BAS 6   BAS 7   ET  =   BAS 6   BAS 8   ET  =   BAS 8   BAS 9   =
BAS 4   BAS 5
AVE <   GAL 2   GAU 1
AVE <   GAL 3   GAU 1
FIN
9,9,4,14
CF  M1  M2  CH  PR  TC  T1  T2  T3
2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,2,6,7,8,1,1,1,1,1,7,8,9,4,C,0,C,C,
2,3,7,8,9,6,4,5,4,5,8,9,6,5,C,0,C,C,
2,2,2,2,2,10,5,10,5,10,5,5,5,3,C,0,C
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,C,C,C,
FIN
```

Enoncé d'un problème de Pfefferkorn

Il y a dans ce problème, au départ, plus de 100 contraintes à satisfaire.

Les heuristiques utilisées par le programme sont les mêmes que dans le problème facile du premier cas.

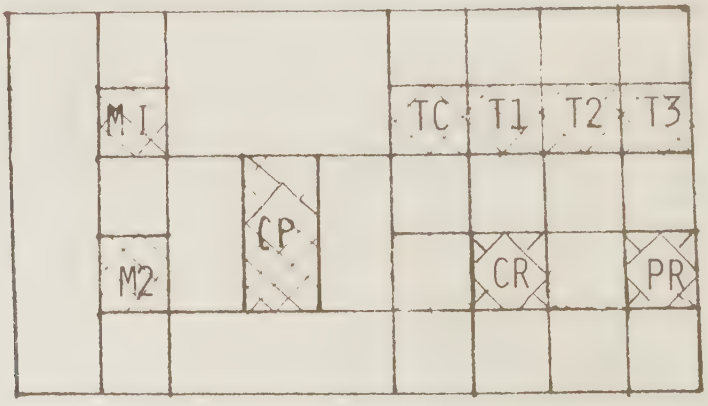
DPS , le programme de Pfefferkorn, regroupe dans une première phase les différents équipements intervenant dans ce problème en "macro-objets" : les d modules mémoire d'une part, le lecteur et l'imprimante d'autre part, enfin le quatre dérouleurs se trouvent ainsi associés. Les contraintes sont elles-même rangées en deux classes : les contraintes "faciles" et les "difficiles". La r solution du nouveau problème concernant les "macro-objets" et les contraintes difficiles conduit par la suite à la solution du problème initial. Cette phas qui constitue donc un plan, est efficace pour résoudre ces problèmes d'archit ture.

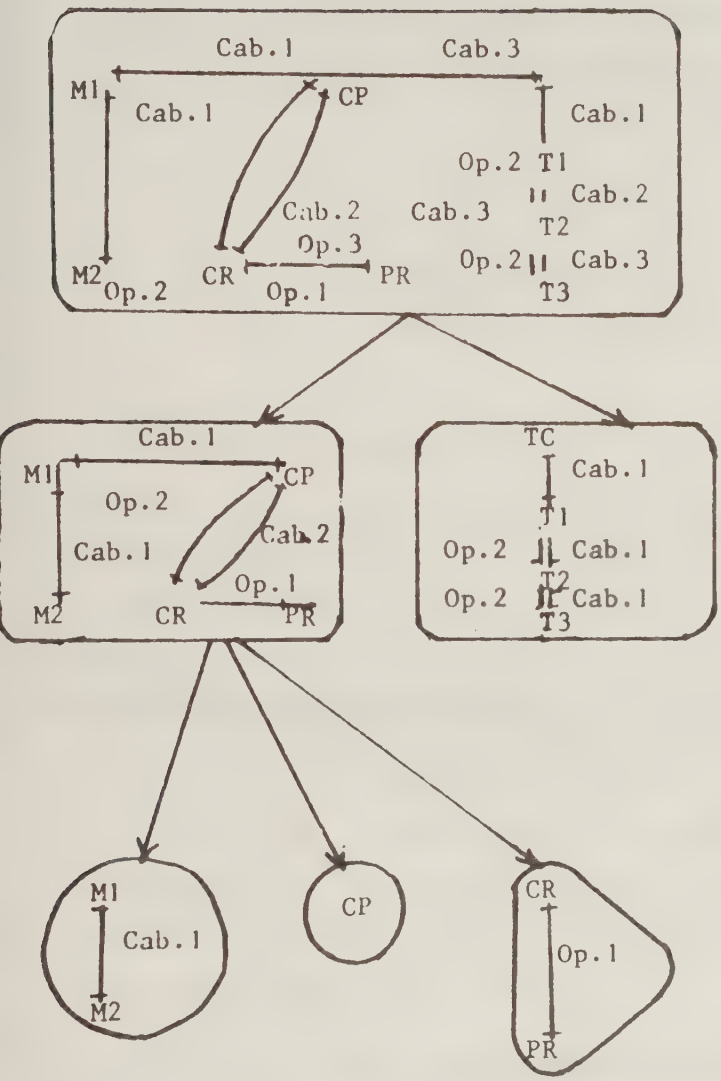
Ce découpage du problème avait également été utilisé pour l'élaboration tomatique des emplois du temps des établissements français (Laurière [1971])

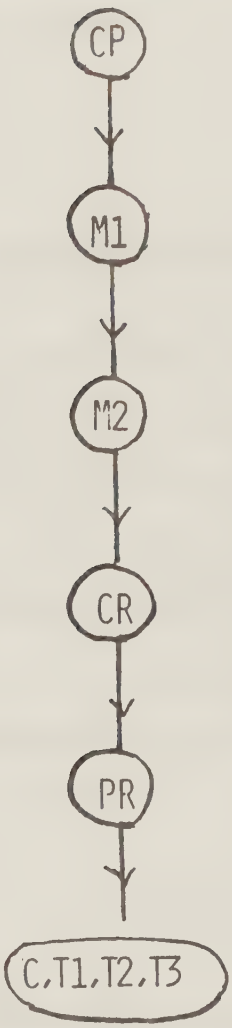
En fait, cette notion capitale de plan, liée à une résolution par arbore cences emboitées est ici naturellement retrouvée par le programme :

Parmi les contraintes "simples" , celles qui ne sont pas simplement mémo sées dans le graphe interne (comme bas 6 = bas 7) , sont analysées en priori notamment avant d'aborder les contraintes "OU" . Par exemple dro 2 < gau 1 entraine gau 1 ≥ 2 . Le nombre de degrés de libertés est ainsi progressiveme réduit a priori.

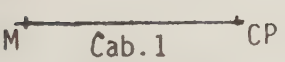
Puis et surtout, les choix se font en considérant les objets qui sont les plus contraints et en faisant intervenir la taille des objets (critère max c ficient): il est clair que les objets qui jouent des rôles symétriques comme quatre dérouleurs ou le lecteur et l'imprimante seront traités immédiatement après l'autre et placés côte à côte par l'intermédiaire des contraintes de c rie. Les macro-objets de Pfefferkorn sont ainsi réobtenus, le programme don la solution de la figure sans retour en arrière en 17 secondes.







ORDRE DES CHOIX DE NOTRE PROGRAMME ET SOLUTION



signifie qu'il est difficile de satisfaire la contrainte de cablage entre M et CP

signifie qu'il est facile de satisfaire la contrainte de passage entre CP et PR

PLAN DE RECHERCHE PAR DSP

IV.6 - PROBLEME DU VOYAGEUR DE COMMERCE.

Trouver le circuit hamiltonien (i.e. passant une fois par tous les sommets) de valuation minimale dans un graphe valué.

Little, Murty, Sweeney, Karel [1963] :

	A	B	C	D	E	F
A		27	43	16	30	26
B	10		19	4	33	28
C	22	15		35	7	2
D	21	16	26		18	18
E	12	46	27	48		5
F	23	5	5	9	5	

```
LITTLE
******
SOI CST N
    ENS E   =   INT 1   N
TRO BIJ AFF >-> E   E   VAL CIR
AVE MIN SIG APP I   E   VAL I   AFF I
FIN
6,
100,27,43,16,30,26,10,100,19,4,33,28,
22,15,100,35,7,2,21,16,26,100,18,18,
12,46,27,48,100,5,23,5,5,9,5,100,
FIN
```

Le programme trouve d'abord ($\varphi 1$) la solution évidente

ABCDEFA (valeur 127.)

(Les arcs $\overrightarrow{AA}$, $\overrightarrow{BB}$, $\overrightarrow{CC}$, $\overrightarrow{DD}$, $\overrightarrow{EE}$, $\overrightarrow{FF}$ sont supprimés une fois pour toutes).

Une solution approchée est ensuite obtenue en choisissant pour chaque sommet l'arc disponible de moindre coût on obtient immédiatement ($\varphi 2$) :

ADCFBEA (valeur 93.)

Le début de ($\varphi 3$) conduit à la réduction des coûts qui se fait en réduisant à chaque fois la rangée dont le coût minimum est minimum sur l'ensemble des rangées , soit ici, dans l'ordre : lignes C, B, E, F, colonne A , lignes D et A, et la matrice irréductible suivante est calculée :

	A	B	C	D	E	F
A		11	27	0	14	10
B	0	—	15	0	29	24
C	14	13	—	33	5	0
D	15	0	10	—	2	2
E	1	41	22	43	—	0
F	12	0	0	4	0	—

avec la borne inférieure $\underline{Z} = 53$.

Les arcs de valuation supérieure à $Z_o - \underline{Z} = 92 - 53 = 39$ sont définitivement supprimés, c'est le cas de $\vec{EB}$ et $\vec{ED}$.

Les choix ont alors lieu, comme dans l'algorithme bien connu de LITTLE et al. [1963] , en prenant à chaque fois l'arc de regret maximal, à ceci près qu'ici le programme applique auparavant le critère :

$$\min_{i \text{ ou } j} \text{degrés liberté} \left(\min \right) \left| \{ j \text{ ou } i \ / \ \exists \vec{ij} \} \right)$$

Le premier choix se fait donc sur E : $fE = F$
et $\vec{FE}$ est supprimé ; une nouvelle réduction a lieu sur la ligne C et $\underline{Z} = 58$. (borne inférieure dans ce contexte).

Plusieurs sommets ont 4 degrés de liberté, le regret maximal (11) correspond à la ligne A : $fA = D$

L'arc $\vec{DA}$ est supprimé. A ne peut être atteint que de B (coût 0) ou de C (coût 10) ; le prochain choix dû au fait que l'on cherche une surjection est :

$$fB = A .$$

La suppression de $\vec{DB}$ implique $\underline{Z} = 60$ puis la solution

ADCEFBA valeur 68 .

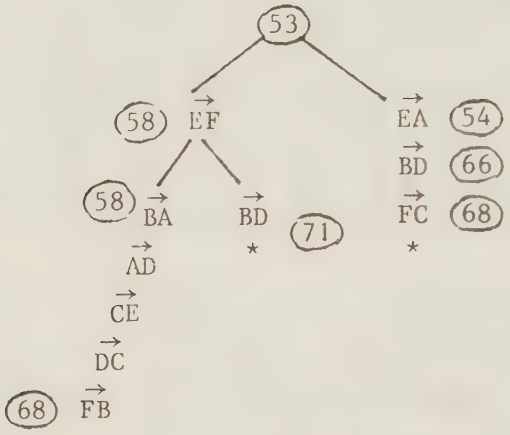
Le programme repart de la racine de l'arborescence après cette première solution de $(\varphi 3)$, en supprimant tous les arcs de valuation supérieure à

$$67 - 53 = 14$$

et en sachant que la dernière solution est au pire à

$$\frac{68-53}{53} = 28\% \text{ de l'optimum}$$

et il décrit l'arborescence finale



montrant que la solution précédente est en fait optimale. (temps total d'exécution une seconde).

Remarque

Le simple fait d'ôter de l'énoncé le mot cir , qui demande à la fonction cherchée de réaliser un circuit transforme le problème en un problème d'affectation:

Trouver une bijection des lignes (machines) , dans les colonnes (emplois) tel que le désagrément total soit minimum.

```
AFFECTATION
**********
SCI CST N
    ENS E   =   INT 1   N
IRC BIJ AFF >-> E   E   VAL
AVE MIN SIG AFF I   E   VAL I   AFF I
FIN
```

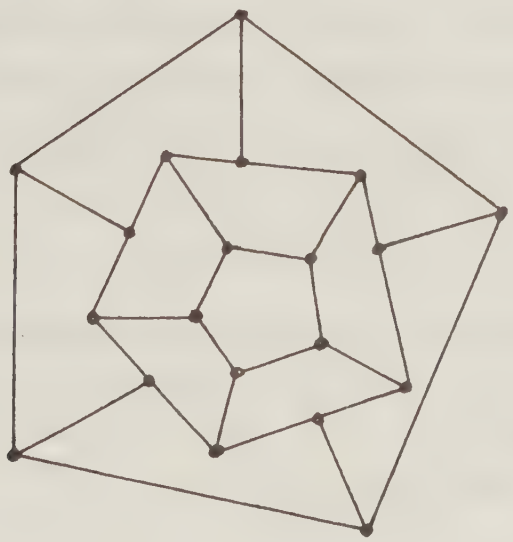
L'approche de la solution est identique, aux implications (évitant les boucles parasites) près, la solution optimale a, bien sûr, une "meilleure" valeur économique : 58 .

Sur des problèmes de taille importante la méthode heuristique décrite en III.7 (cas Z2) permet en outre d'obtenir rapidement des solutions approchées.

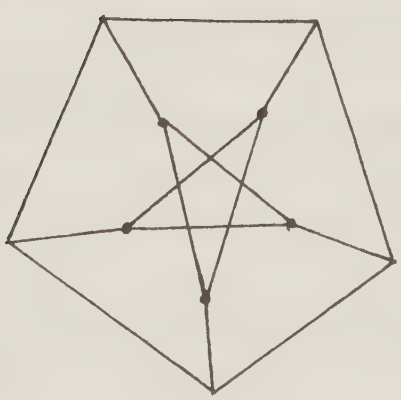
Problème d'existence d'un circuit hamiltonien dans un graphe :

Ce problème a été étudié depuis longtemps en théorie des graphes.

Il permet en outre de modéliser certains problèmes d'ordonnancements disjonctifs.



Graphe d'Hamilton (dodécaèdre) (un circuit hamiltonien obtenu en 2 s)



Graphe de J.Petersen (pas de circuit hamiltonien (1 s), bien que le graphe soit 3-chromatique)

Un problème d'ordonnancement lié à l'existence de circuit :

N produits doivent être fabriqués sur une même machine ; suivant l'ordre dans lesquels ils passent il y a lieu ou non d'arrêter la machine pour la nettoyer ou bien échanger une de ses pièces : on cherche l'ordre optimal i.e. minimisant le nombre d'arrêts connaissant la matrice d'éléments booléens

$$m(i,j) = \begin{cases} 1 & \text{si j suit immédiatement i pas d'arrêt} \\ 0 & \text{" arrêt} \end{cases}$$

(cf. par exemple Selby [1970] Christofides [1973]).

Ce problème peut être résolu par approximations successives: on ajoute au graphe orienté de matrice d'incidence $\mathcal{M}$ p sommets liés dans les deux sens à tous les autres; on cherche alors le plus petit p à partir duquel il existe dans ce graphe un circuit hamiltonien.

Exemple :

$\mathcal{M} =$

	1	2	3	4	5
1		1	1		1
2				1	
3	1	1		1	
4	1				
5	1	1			

Il n'y a pas de circuit hamiltonien, mais avec un arrêt entre 4 et 5 , on obtient un ordonnancement optimal : 1 3 2 4 5 1 .

Le problème est donné comme pour le voyageur de commerce mais avec l'option SUC.

Le programme prend 11 secondes pour trouver un circuit hamiltonien dans un graphe aléatoire de 150 sommets avec une moyenne de 5 éléments à 1 par ligne dans m(i,j) , avec p fixé arbitrairement à 4 . Il lui faut 177 secondes pour démontrer qu'il n'existe pas de solution avec p=3 .

Des problèmes de théorie des graphes se ramènent également à la recherche d'un circuit hamiltonien, en particulier la détermination du nombre de chemins élémentaires qui partitionnent l'ensemble des arcs d'un graphe complet symétrique ce problème très combinatoire et il n'a pas été possible de traiter les cas intéressants ($n \geq 9$). [Chaty, Chein, Martin, Petolla (1975)]

Problèmes de tournées avec contraintes :

Il s'agit d'une généralisation du problème du voyageur de commerce : étant donné, de la même façon, un graphe valué, il faut joindre tous les sommets par cette fois des circuits en nombre p, qui ne se recoupent qu'en un dépôt central soit A, en minimisant la somme des distances parcourues; p est donné et correspond au nombre de camions nécessaires à la tournée et reste dans les problèmes réels inférieur à la dizaine. Chaque camion possède en outre une capacité limitée L ; il doit déposer (ou prendre) en chaque sommet i une charge c_i donnée sans dépasser sa capacité totale.

On a donc une contrainte supplémentaire qui s'écrit :

AVE <= SIG ET APP I E CHE I A C I L ($\sum_{i \in E} c(i) \leq L$)

$\exists$ chemin IA

Le nombre p est choisi égal à $\left\lceil \frac{\sum_{i \in E} c(i)}{L} \right\rceil$.

On sait que ce type de contrainte est interprété par le programme et intervient sous forme de test.

Il peut en fait s'ajouter à la contrainte de capacité en charge des contraintes de même type concernant par exemple le volume des camions.

Le programme a été testé sur des jeux de données (voyageur et tournées) plus importants. Dans la pratique, seules sont demandées de bonnes solutions approchées, aussi le programme est-il arrêté dans $\varphi 3$ après dix solutions ou au bout de 2 minutes et donne la distance à l'optimum. Dès que les données comportent plus de 10 sommets la méthode de transformation des coûts (III.7.Z2) est lancée. Elle a conduit, dans environ un cas sur deux, à une solution meilleure que la première obtenue dans $\varphi 3$ par réductions et regrets (mais huit problèmes seulement ont été testés; ils sont tous construits à partir des distances sur les 48 villes des Etats Unis, Held et Karp [1962]) . En outre, dans la phase 3 pour ces mêmes problèmes le choix de type (4) (III.4) :

$\forall j \qquad c(i,j) \leq \beta_i$

supprime un arc sur deux sur la ligne i après les réductions; rendant le problème plus contraint il fait jouer le critère

MIN degré libertés

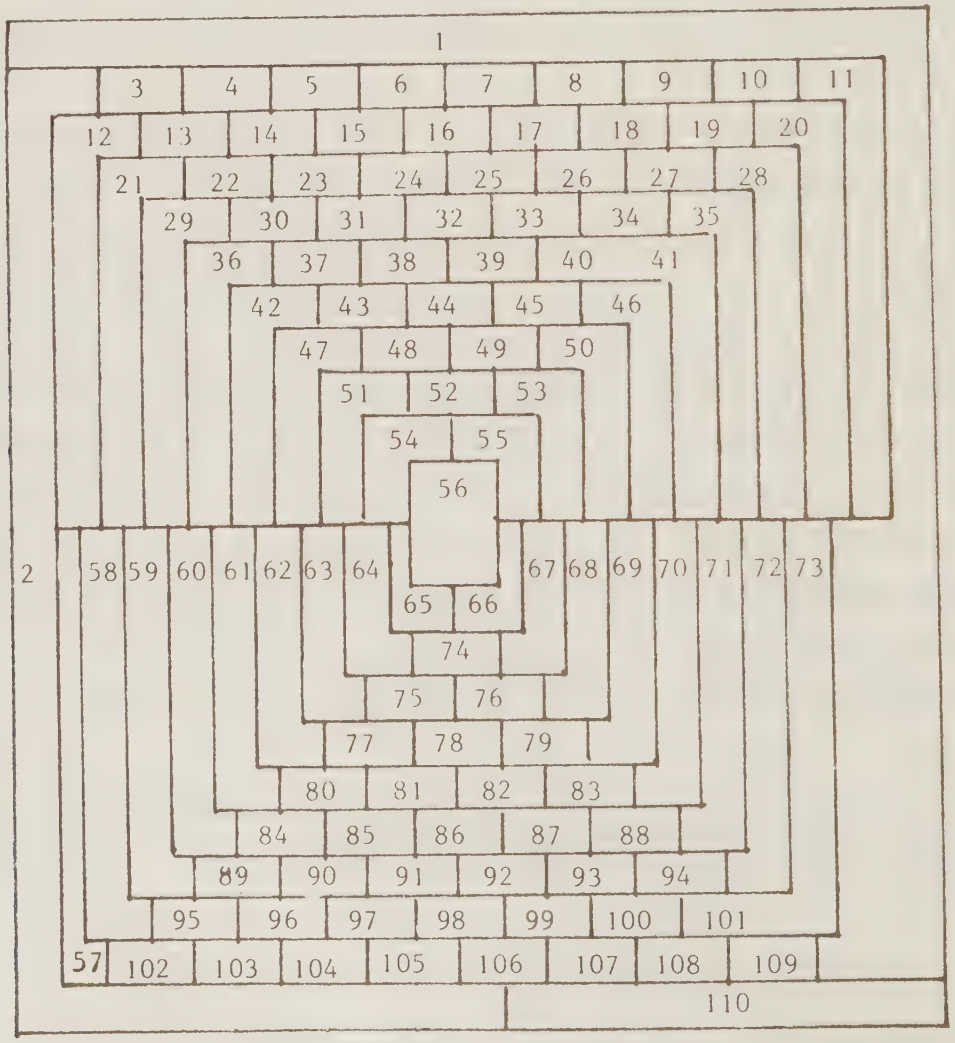
et permet d'obtenir rapidement des solutions. La borne inférieure $\underline{Z}$ est malheureusement très optimiste si bien que la distance théorique à l'optimum après 2 minutes d'exécution n'est jamais descendue en dessous de 16% , alors qu'il y a tout lieu de penser que les solutions étaient en fait à moins de 2 ou 3% de cet optimum. Un circuit hamiltonien de 11468 miles est notamment obtenu dès $\varphi 2$ pour le graphe complet des 48 villes .

Rappellons que le problème du cavalier sur l'échiquier 8×8 est résolu en 4 secondes ; la méthode de Held et Karp [1971] , qui est donc mal adaptée à ce cas, demande 6,9 minutes (IBM 360/91).

IV.7 - PROBLEMES DE COLORATION.

Dans le numéro d'avril ... 1975 du Scientific American, Martin Gardner proposait un graphe planaire (cf fig.) de 110 régions, qui devait infirmer la conjecture des quatre couleurs.

Le programme colorie ce graphe en quatre couleurs avec seulement deux remontées dans l'arborescence en 12 secondes.



L'énoncé ALICE est :

```
COLORATION
**********
SOI CST N   F
    ENS SMT =   INT 1   N
        COL =   INT 1   F
TRC FCN F   >-> SMT COU DIS
FIN
110,4,
2,3,4,5,6,7,8,9,10,11,20,73,109,110,0,0,0
3,12,57,102,103,104,105,106,110,0,0,0,0,0
4,12,13,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5,13,14,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6,14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
7,15,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8,16,17,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9,17,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10,18,19,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

(les données ont ici été coupées après les successeurs des six premiers sommets)

Les critères sont dans l'ordre :

POUR I MIN successeurs
MAX disjonctions

POUR J MIN $|\{i/fi=j\}|$
MIN antécédents

on choisit donc le sommet pour lequel il reste le moins de couleurs et qui est en disjonction avec le plus grand nombre de sommets; ce sommet est colorié avec la couleur la moins utilisée (coloration équilibrée) et en cas d'égalité avec celle qui permet de colorier le moins de sommets non colorés.

La détection de la clique des quatre sommets 1, 10, 11, 20 permet immédiatement de conclure l'optimalité des quatre couleurs.

Problème de planning de faculté.

Connaissant les inscriptions des étudiants aux divers examens et le nombre d'amphithéâtres disponibles pour chaque demi-journée, il s'agit de prévoir, en minimisant la durée totale, les examens d'une session universitaire.

Ce problème peut être considéré comme un problème de coloration où les couleurs correspondent aux demi-journées. Des contraintes additionnelles, imposant des dates au plus tôt et au plus tard, pour chaque épreuve peuvent être prises en compte (option suc). Un étudiant au moins inscrit à deux examens, entraine

une disjonction entre ces examens (option dis).

Enfin si l'on veut distinguer, par le nombre d'étudiants qu'ils peuvent recevoir, les amphithéâtres, une deuxième coloration, où les couleurs correspondent cette fois aux amphis, est à prévoir pour chaque demi-journée : seuls certains amphis conviennent alors (option suc) et le nombre d'amphis maximal total est imposé.

Ce problème est résolu pour un problème réel de 186 examens, 27 amphis en 16 secondes, la solution optimale utilise 21 périodes.

La démonstration de l'optimalité n'est pas toujours simple puisque l'on sait qu'il est possible de construire un graphe dont l'écart entre le nombre chromatique et la cardinalité maximale d'une clique est plus grand que tout nombre donné a priori.

Aide à la conception en architecture :

C. Roucairol et A. David [1972] , pour aider à bien prendre en considération toutes les contraintes qui interviennent par exemple dans l'aménagement d'une chambre dans un hôpital, cherchent dans le graphe des contraintes un recouvrement par des cliques maximales. (Deux contraintes sont liées par une arête si et seulement si elles font intervenir le même élément de construction). On a donc un problème de coloriage dans le graphe complémentaire. Un exemple faisant intervenir 40 contraintes est traité en 17 secondes.

Ludot [1969] traitait un problème similaire par la programmation mathématique.

Pour les graphes n° 3 et 4 de la dernière référence, il faut au programme respectivement 38 et 71 secondes au lieu de 256s et 822s (CDC 6600).

Signalons en outre que les algorithmes proposés dans la littérature ne donnent que des solutions approchées Welsh et Powell [1968] Wood [1970] ou encore ne peuvent traiter , à cause de la place mémoire qu'ils requièrent, des graphes de plus d'une cinquantaine de sommets : Christofides [1971] Corneil et Graham [1973] .

Notons à ce propos que l'évaluation d'une borne de la place mémoire pour un algorithme , qui indique directement si un problème passe ou non sur les ordinateurs actuels, apparait pratiquement comme plus importante que la borne en temps de calcul, habituellement étudiée en complexité des algorithmes.

nombre de sommets	40	99	99	110*	130	150+	150*
temps de calcul (secondes)	17	38	71	12	98	162	18

* : graphes planaires.

\+ : moyenne sur 10 graphes aléatoires.

Résultats sur les problèmes de coloration.

Couplages maximaux de cardinalité minimum.

Soit à trouver dans l'ensemble des arêtes du cube unité de $\mathbb{R}^n$ un sous-ensemble d'arêtes :

- non adjacentes deux à deux (couplage)
- maximal pour cette propriété i.e. auquel on ne peut rajouter aucune nouvelle arête.
- de cardinalité minimum.

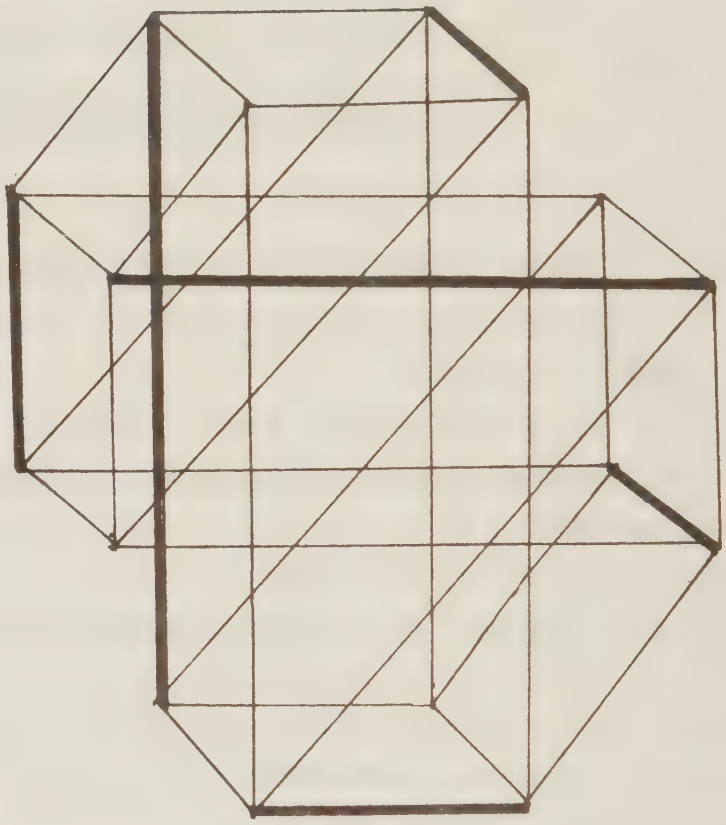
Ce problème est notamment étudié par Forcade [1973] .

Pour $n \leq 5$ on obtient par la théorie la cardinalité optimale $3 \times 2^{n-3}$.
Pour ces valeurs le programme a exhibé de tels couplages optimaux en 5 secondes pour n=4 , 4 minutes pour n=5 .

```
CCLPLACE
********
SCI CST N
    ENS E   =   INT 1   N
        F   =   INT 1   1
IRC RES C   >-> E   F   DIS DMI DMA MAX
```

sous ensemble
non adjacences
maximalité
cardinalité

```
FIN
32,
  2, 3, 4, 5, 6, 7,
  3, 4, 8, 9,1C, C,
  4,13,14,15, C, C,
 21,22,23, 0, 0, C,
  6, 7, 8,11,12, C,
  7,13,16,17, C, C,
 21,24,25, C, C, C,
  9,1C,11,12, C, C,
 1C,14,18,19, C, C,
 22,26,27, 0, C, C,
 12,16,18,2C, C, C,
 24,26,28, C, C, C,
 14,15,16,17, C, C,
 15,18,19, C, C, C,
 23,29,3C, 0, C, C,
 17,18,2C, C, C, C,
 25,29,31, 0, C, C,
 19,2C, C, C, C, C,
 27,3C,32, C, C, C,
 28,31,32, C, C, C,
 22,23,24,25, C, C,
 23,26,27, C, C, C,
 29,3C, C, C, C, C,
 25,26,28, C, C, C,
 29,31, C, C, C, C,
 27,28, C, C, C, C,
 3C,32, C, C, C, C,
 31,32, C, C, C, C,
 3C,31, C, C, C, C,
 32, C, C, C, C, C,
 32, C, C, C, C, C,
  C, C, C, C, C, C,
6,
6,
FIN
```



Couplage optimal (n=4)

IV.8 - UN PROBLEME DE PAPETERIES.

Il faut fabriquer des quantités q_i de papier différents $(i=1,\ldots,n)$, sur des machines données $(j=1,\ldots,m)$.

Le coût de fabrication, pour chaque machine et chaque sorte de papier, est connu et, de façon à éviter l'arrêt des machines pour réglage, un papier ne peut être fabriqué qu'entièrement sur le même appareil.

On demande l'ordonnancement optimal.

Exemple : Sandi [1974]

	3	5	10	1	2	3	5	3	8	2	← quantité demandée q_i du produit i
1	13	9	1	3	14	15	7	9	15	8	9
2	13	9	20	6	17	7	9	16	17	1	10 ← capacité a_j de la machine j
3	18	15	11	14	13	8	8	12	17	12	20
4	5	4	19	2	6	15	19	4	15	20	3
j ↑ / i →	1	2	3	4	5	6	7	8	9	10	

← coûts c_{ij}

il s'agit donc de trouver pour tout i de 1 à n une affectation f(i) telle que :

$$\min \sum_{i=1\ldots n} c(i,f(i)) \qquad (1)$$

$$\forall j \sum_{\substack{i=1\ldots n \\ f(i)=j}} q_i \leq a_j \qquad (2)$$

Les types de contraintes sont classiques et Ross et Soland [1975] proposent une variante de ce problème justiciable du même type de méthode.

(2) $\Rightarrow$ $f(3)\neq 1 \quad f(2),f(3),f(7),f(9)\neq 4$.

Les critères retenus sont

min degré liberté dans $\varphi 1$

min coût dans $\varphi 2$

et dans la phase 3 , la contrainte (1) entraine une réduction des coûts avec

$$\underset{\text{max}}{Z}_{\text{regret}} = 67 \qquad \text{et le critère.}$$

Le programme obtient la solution optimale

i	1	2	3	4	5	6	7	8	9	10
f(i)	4	2	3	1	3	2	1	1	3	1

de valeur 82.

Problème de localisation d'entrepôts.

Le problème est semblable au précédent mais des frais fixes s'ajoutent à la fonction économique :

Disposant d'un nombre fini d'emplacements $j=1...n$ où il est possible de construire des entrepôts pour satisfaire la demande de clients $i=1...m$, déterminer le nombre et l'emplacement des entrepôts pour minimiser les coûts de construction et de transports. Ce problème et sa formulation en programmation mathématique sont dus à Balinski [1964].

Exemple Spielberg [1969] :

			i=	1	2	3	4	5	6	7	8	9	10
j=	1	3		4	×	5	×	9	13	5	6	26	×
	2	3		×	1	×	2	11	21	×	15	9	8
	3	4		1	2	11	21	15	×	×	9	×	8
	4	4		1	2	11	21	15	×	×	9	×	8
	5	5		1	×	2	11	21	15	9	×	8	×
	6	6		4	×	29	×	×	20	×	11	15	×
	7	7		×	4	×	29	20	×	11	15	×	×
	8	7		×	×	1	×	×	2	11	×	21	15
	9	7		4	29	20	11	15	22	3	4	29	20
	10	8		12	13	5	19	21	27	8	4	29	20
		coûts (fixes) de construction g_j		coûts de transport c_{ij}									

Une croix en ij spécifie que le client i ne peut être livré depuis le $j^{ème}$ dépôt.

La fonction économique s'écrit :

$$\sum_{\forall j \mid f(i)=j} \; \sum_{i=1..m} g(j) \; + \; \sum_{i=1..m} c(i,f(i))$$

On peut donc, d'entrée de jeu, supprimer dans ce problème tout arc $\vec{ij}$ tel que

$$\forall k \;,\; k \neq j \qquad c_{ik} + g_k < c_{ij}$$

et cette contrainte a été ajouté à l'énoncé ;

```
SCI CST N   M
    ENS EI  =   INT 1   M
        EJ  =   INT 1   N
    CCE G   SUR EJ
IRC FCN F   >-> EI  EJ  SUC VAL
AVE CCS APP I   EI  CCS APP J   EJ  ==> EXI APP K   EJ  ET  ¬=  K   J   <   +
VAL I   K   G   K   VAL I   J   ¬=  F   I   J
AVE MIN +   SIG ET  APP J   EJ  EXI APP I   EI  =   F   I   J   G   J   SIG APP
I   EI  VAL I   F   I
FIN
```

Le programme supprime alors 39 arcs et obtient par des descentes directes des solutions en phases 1 et 2 . La phase 3 commence ici encore par une réduction donnant $\underline{Z} = 39$ et bien que le programme ne sache pas augmenter cette borne inférieure par manipulation des $g(j)$, il obtient alors la solution

i	1	2	3	4	5	6	7	8	9	10
f(i)	1	2	8	2	1	8	1	1	2	2

de valeur 60 en utilisant les critères

pour i : min degrés libertés
max regret

pour j : max $|\{i/f(i)=j\}|$

Cette solution est optimale car on a les affectations forcées dès le début de $\varphi 3$:

$$f(4) = 2 \quad \text{et} \quad f(6) = 8 \quad \text{donc} \quad \underline{Z} = 49$$

puis $f(5) = 0$ OU $f(5) = 1$ et la démonstration complète est achevée en 5 secondes. Le programme a résolu plusieurs problèmes 20×40 en une trentaine de secondes en moyenne.

IV.9 - LE PROBLEME DES ROULEMENTS A LA SNCF :

Partant d'horaires des trains imposés, il s'agit de mettre sur ces trains des conducteurs en respectant les contraintes syndicales et en minimisant le temps total d'embauche du personnel.

Le réseau est séparé en six districts. A l'intérieur de chacun d'eux, les services de conduite des trains sont regroupés en tournées : une tournée part d'un dépôt et y revient après un ou plusieurs jours. L'ensemble des tournées est donné. Les conducteurs assurent les tournées, l'une après l'autre, par roulement, dans un ordre à déterminer. Le problème concerne une durée d'activité totale de plusieurs mois et le service doit être équilibré tout au long de cette période.

Entre deux tournées quelconques se trouve un repos ; ce repos peut être de trois types :

repos normal : minimum 14^H , maximum 38^H ;
repos périodique simple : repos normal plus un jour complet de repos à domicile ;
repos périodique double : deux jours complets et consécutifs de repos plus un repos normal.

Les repos périodiques ne sont comptés qu'à partir de 20 heures et jusqu'à 6 heures dans les journées correspondantes; ils doivent si possible correspondre aux fins de semaines civiles.

Un ensemble de tournées se succédant sans repos périodique est appelé grande période de travail ou GPT et doit comporter au moins deux et au plus six jours de travail.

En outre ,

- une GPT de six jours doit obligatoirement être suivie d'un repos périodique double.
- il faut minimiser les temps blancs, c'est-à-dire les temps entre travail et repos.
- il faut minimiser le nombre total de repos périodiques tout en respectant un minimum de 36 repos périodiques doubles sur l'année.
- la longueur parcourue doit être distribuée de manière équitable le long d'une G.P.T.

Méthodes de résolution du problème.

a) Historiquement J.M. Pla (1972) a d'abord proposé un premier modèle opérationnel décomposé en deux phases. La recherche d'un flot de coût minimum dans le graphe des tournées par la méthode "out-of-kilter" , est suivie d'une procédure arborescente qui travaille sur le multigraphe dessiné par ce flot.

La seconde phase ne parvient cependant pas toujours à établir un roulement complet.

b) Par la suite, le problème a été abordé comme les problèmes semblables qui se posent dans d'autres compagnies de transports par l'intermédiaire d'un générateur et d'un modèle de programmation en nombres entiers comportant essentiellement des contraintes de partitionnement.

c) En fait si l'on considère le graphe où les sommets sont les tournées et où les arcs indiquent les successeurs possibles, alors on cherche sur ce graphe, ici un circuit hamiltonien - un ensemble de chemins disjoints dans le problème RATP du réseau routier - qui doit vérifier les contraintes et satisfaire les objectifs.

(départ de j > arrivée de i + 14^H)

Dans l'impossibilité où est aujourd'hui la recherche opérationnelle de garantir ici l'optimum optimorum, une méthode directe, sans modélisation mathématique trop abstraite éloignant du monde réel, est préférable. Une résolution approchée de ce problème des roulements a été demandée au programme utilisant la présentation c).

Dans les méthodes a) et b) il est nécessaire en effet de faire des hypothèses et de négliger certains points difficiles à modéliser pour parvenir à une taille de programme mathématique que les codes actuels peuvent aborder :

il faudrait ensuite remettre en question le flot de la première phase pour la méthode a), prendre en compte toutes les partitions données par le générateur b).

Cela est concrètement interdit par le temps calcul. De plus, la demande des utilisateurs se fait plus exigeante vis à vis des informaticiens : les problèmes soumis deviennent plus importants et leur taille augmentant, leur nature même change, au point qu'il est impossible d'espérer appréhender de façon rigoureuse les nouveaux problèmes à l'aide des modélisations habituelles.

Une méthode, proche de la méthode manuelle , telle que celle utilisée par le programme conduit alors à des meilleures performances :

Le temps calcul du programme ne représente pas le quart de celui demandé pour résoudre (résolution en continu, après quoi il reste encore beaucoup à faire) dans la formulation b , le programme linéaire associé.

Enfin, l'obtention d'une borne inférieure par le processus de réduction des coûts (IV.6) permet le contrôle de la distance à l'optimum de la valeur minimale du temps blanc. Ainsi la solution du graphique a la valeur 4760 avec une borne inférieure de 4651 et se trouve à 3 pour cent au plus au-dessus de l'optimum.

Résultats

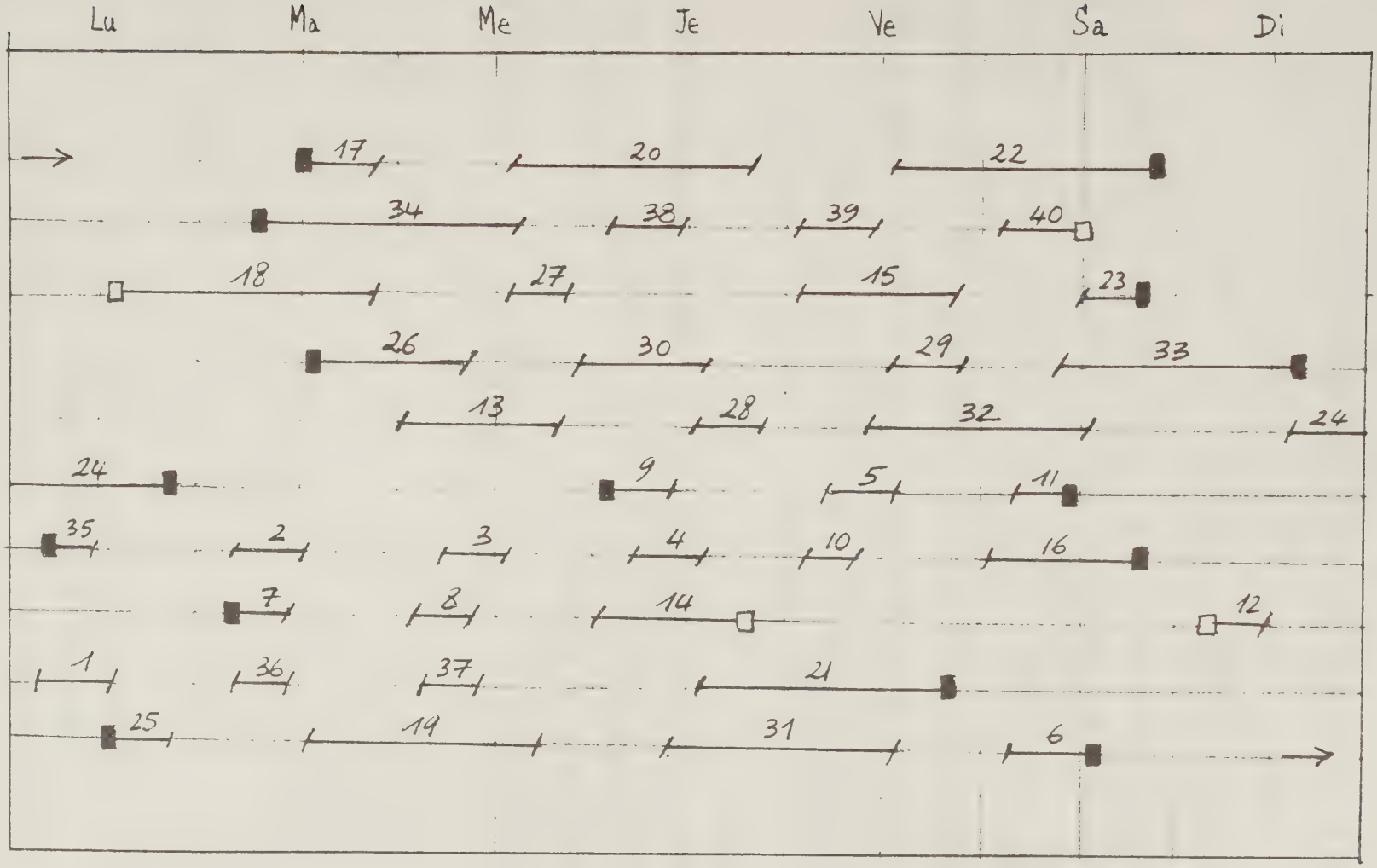
Problème	Nb de tournées	temps CPU (1ère solution de φ_3)
Angoulême	40	17
Calais	40	14
Chaumont	53	31

Données du problème des roulements pour le district Angoulême.

1 à 6	0602	1221	0	111111C	C619
7 à 12	0331	1014	0	C111111	C6C9
13 à 15	CC29	2202	0	CC111CC	1225
16	CC29	2212	0	CCCCC1C	1118
17	12C0	22C0	0	C1CCCCC	C8CC
18 à 22	1215	2036	1	11111CC	1422
23	1215	1910	0	CCCCC1C	C552
24	123C	2036	1	CCCCCC1	1619
25 à 29	1233	2CC0	0	11111CC	C727
30 à 33	1138	1320	1	CC1111C	1C55
34	1054	1320	1	C1CCCCC	122C
35 à 40	C415	1151	0	111111C	C713

numéro de la tournée — départ des tournées — arrivée des tournées — nombre de passage à minuit — amplitude

jours de départ du lundi au dimanche
autant d'éléments à 1 que de tournées par ligne



■ extrémité d'un repos double

□ extrémité d'un repos simple

SOLUTION DU PROBLEME DE ROULEMENTS PRECEDENT (Angoulême)

Le programme a été écrit en PL1 sur 1' IBM 370-168 du C.N.R.S. à Orsay (C.I.R.C.E.).

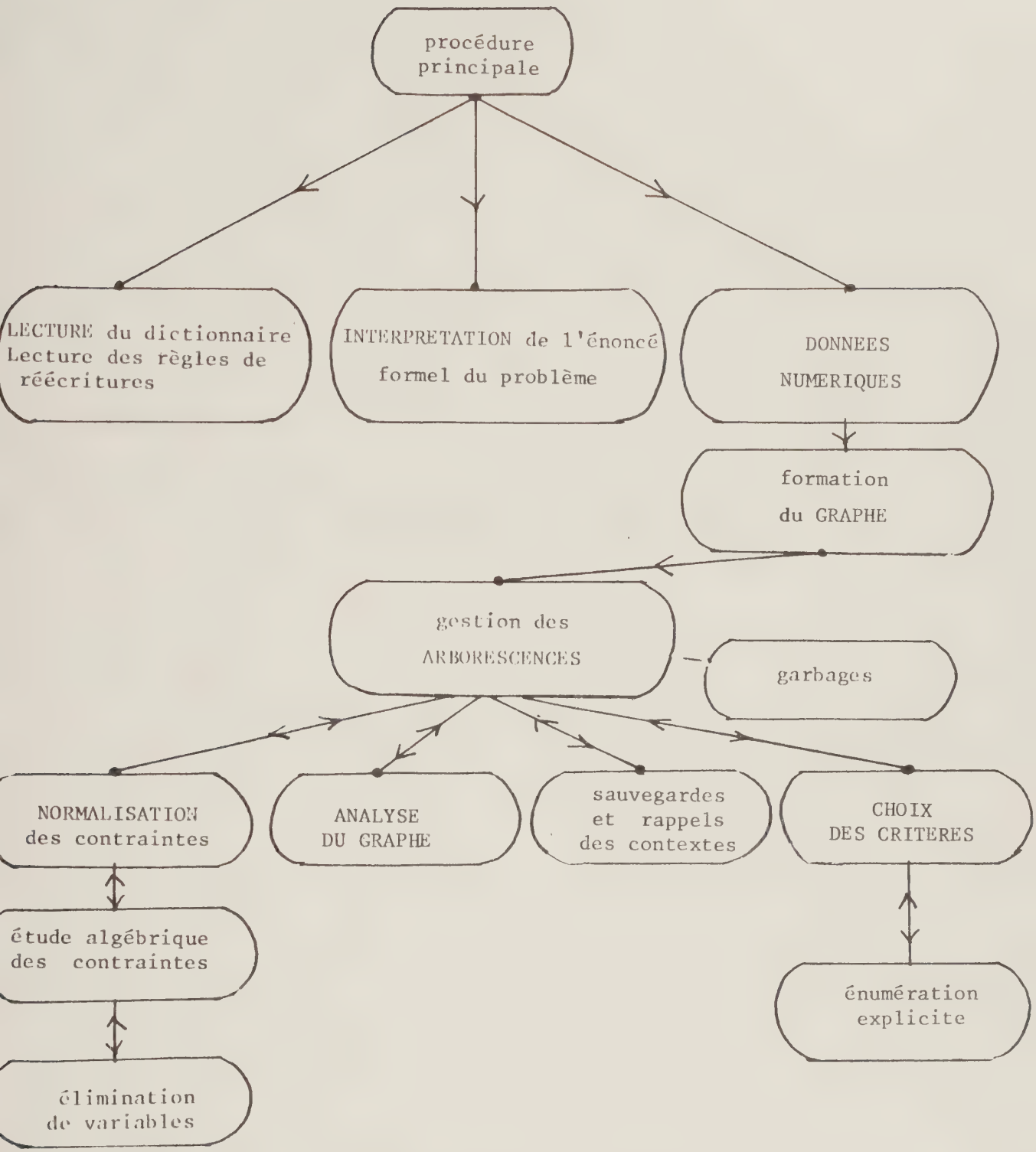
Il comporte environ 7000 instructions.

Aucune possibilité particulière à PL1 n'a finalement été employée, la récursivité par exemple coûtant beaucoup trop cher en temps calcul.

Plus de deux cents problèmes ont à ce jour été résolus.

La place mémoire importante occupée par le code seul a été réduite de 460 K octets à 120 en utilisant une structure d'overlay de façon à pouvoir le plus souvent possible exécuter le programme sous le système conversationnel en temps partagé (TSO).

Trois "garbage collectors" récupèrent la place mémoire en cours d'exécution : l'un sur la liste des cliques, le second sur la liste des arcs, le dernier sur les piles d'expressions en polonaise.



SCHEMA GENERAL DU PROGRAMME

ET STRUCTURE D'OVERLAY EN MACHINE

Les modules d'un même niveau ne sont jamais en mémoire en même temps.

REFLEXIONS ET CONCLUSION

- Un certain nombre de modifications, qui n'affectent pas la démarche fondamentale du programme, pourraient améliorer de façon sensible les performances.

Simplification du programme pour traiter un problème particulier.

Pour un problème donné, un nombre d'instructions, qui peut être élevé par rapport à l'ensemble du programme, est inutile a priori dès la lecture de l'énoncé : par exemple le problème ne comporte pas d'optimisation, ou/et il n'y aucune contrainte de type algébrique (et le programme n'en crée aucune), ou/et les contraintes sont toutes linéaires en fonction des inconnues.

L'économie réalisée alors est essentiellement due à la suppression des renseignements connus à l'avance :

ainsi la mémorisation et la gestion des valeurs maximale et minimale de toute variable , utiles dans le cas général, sont superflues quand les variables sont booléennes; de même une version très simplifiée de l'algorithme d'unification suffit quand toutes les contraintes sont linéaires.

La représentation des expressions sous forme polonaise a été choisie pour d'une part, ne pas utiliser trop de place mémoire, et d'autre part, ne pas poser de problèmes de programmation pour la partie manipulation formelle algorithmique. Il s'avère maintenant que pour certains énoncés où plusieurs milliers d'unifications sont nécessaires (programmation en nombres entiers essentiellement), les techniques classiques sont ici à revoir.

L'introduction d'opérateurs (+,*,OU) variaires, (i.e. chaque opérateur est suivi d'un nombre quelconque de termes) règle la question de l'associativité de ces opérateurs. Elle supprime du même coup les règles de réécriture correspondantes. Les règles de simplification utilisant en plus la commutativité, l'existence d'éléments neutres et d'inverses gagneraient à être également prévues au niveau du programme, l'unification n'étant réservée qu'à quelques cas précis : négation d'une expression, interprétation lors de la lecture de l'énoncé des contraintes ou de fonctions économiques de types spéciaux.

Modification du programme par lui-même.

Etant donnée une contrainte connue

(1) $$f\ i = j \Rightarrow \forall k \in \text{pre}\ j \qquad f\ k = j$$

(si l'image de i est j , tous les prédécesseurs de j ont également j pour image) qui intervient par exemple dans les problèmes de partitionnement, il est simple, au lieu de l'interpréter, de la traduire directement au niveau des instructions.

Il s'agit d'ajouter au sous-programme d'affectation, qui fixe effectivement j pour image de i , une boucle sur les prédécesseurs de l'image.

Un premier passage donne alors automatiquement les instructions correspondantes et l'endroit du programme où il convient de les intégrer. Dans un deuxième passage, la contrainte est supprimée et l'on fait exécuter un programme spécialisé. Cette modification a été faite pour la contrainte (1) et la contrainte (2) du même problème de partitionnement (IV.1).

Un programme du type de celui écrit par A. Adam [1972] et dont la sortie est donc une série d'instructions permettrait la prise en compte de contraintes plus générales, par exemple

$$f\,i = j \Rightarrow \forall k \in E \quad k \neq i \Rightarrow f\,k \neq j + i - k$$

Extension du langage.

Parmi les problèmes concrets traités par les chercheurs opérationnels, se distingue la classe des problèmes d'organisation des emplois du temps. Tous ces problèmes sont difficiles et mal résolus car leur caractéristique commune est de faire intervenir des conditions sur les hommes eux-mêmes, et les exigences humaines ne se laissent pas modéliser très facilement.

L'organisation des équipages d'avions, de bus ou de trains, de la rotation des équipes de trois huit dans le travail en continu et d'autres problèmes d'emploi du temps font cependant appel à un certain nombre de notions communes. Si la formulation de ces problèmes est très lourde en langage mathématique habituel, quelques abréviations seraient fort utiles :

"équilibrer" telle fonction par rapport à tel vecteur de coûts.
"être égal en moyenne sur telle période à" ,
"minimiser les temps blancs" .

- Certaines modifications plus fondamentales sont à envisager.

Les échecs dus à la généralité des problèmes qu'il est possible de poser en ALICE sont en effet nombreux.

Si ces cas d'échecs ne sont toujours provoqués que par la limitation en temps - il est théoriquement toujours possible d'obtenir la solution en un temps fini d'un problème ne faisant intervenir que des ensembles finis - les causes d'échecs sont de natures diverses :

le programme manque indéniablement dans certains cas d'outils - théorèmes et algorithmes - qui donneraient immédiatement la solution de sous problèmes. (A)

le programme ne dispose pas de bonnes fonctions d'évaluation et l'optimalité peut être trop longue à démontrer. Ce phénomène est général dans les procédures arborescentes et les résultats théoriques manquent pour systématiser la construction de fonction d'évaluation réalisant un bon compromis entre la complexité et la justesse de ces évaluations. (B)

le programme n'a pas une bonne représentation du problème : ceci est particulièrement net dans les cas où notre sens visuel nous dicte de façon immédiate des heuristiques ou des raisonnements. (C) : problèmes d'architecture (IV.5) ou puzzles dans l'espace (instant insanity (IV.1) ou pentaminoes).

A - INTRODUCTION DANS LE PROGRAMME D'ALGORITHMES SPECIFIQUES.

J'ai volontairement privé le programme d'algorithmes jugés fondamentaux en recherche opérationnelle comme le simplexe ou l'algorithme de couplage avec coûts d'Edmonds.

Si le premier ne m'apparait pas important ici, d'autant qu'il est de nature exponentielle, le second, qui s'adapte bien au graphe représentatif interne, qui est polynomial, qui est assez général puisque la notion de couplage apparait comme sous-problème dans de nombreux énoncés combinatoires, permettrait, dans environ 20% des cas traités, un gain notable à l'exécution.

Le théorème de König Hall, utilisé dans Laurière [1971], oblige à étudier les 2^n parties d'un ensemble de n éléments et n'est pas directement programmable; il fournit cependant des résultats fondamentaux dans les cas critiques où un couplage ne peut exister et il conviendrait d'en programmer une forme affaiblie, et un travail théorique établissant une condition nécessaire simplifiée reste à faire.

B - APPEL DU PROGRAMME PAR LUI-MÊME ET AMELIORATION DES FONCTIONS D'EVALUATIONS.

La technique appelée "relaxation" en recherche opérationnelle ne peut être à l'heure actuelle utilisée dans tous les cas; partant d'un problème d'optimisation sous les contraintes $\{\mathcal{C}_i\}_{i \in K}$ la relaxation consiste à oublier un sous-ensemble $\{\mathcal{C}_i\}_{i \in J}$ avec $J \subset K$ de contraintes et à résoudre le nouveau problème: l'optimum du premier problème est par hypothèse "moins bon" que celui du second et l'on obtient ainsi une évaluation de l'optimum cherché.

L'ensemble J est facile à détecter : il faut se ramener à un problème connu et J doit être le plus petit possible.

Pour le programme, cela reviendrait à avoir en mémoire les problèmes qu'il "résoud bien" et à s'appeler lui-même en chaque sommet de l'arborescence, en chaque contexte, pour calculer une meilleure évaluation du contexte.

C - LE PROBLEME DE LA REPRESENTATION.

Alors qu'un programme est quelque chose de relativement figé, nous disposons nous, d'une grande faculté d'adaptation. Et c'est une des recherches fondamentale en intelligence artificielle que de dégager les outils et les heuristiques qui nous permettent une telle attitude.

Parmi les outils dont nous disposons, l'oeil, qui permet une vision globale, et la main, qui peut faire pour lui de petits dessins et créer de nouveaux objets, nous sont de précieux auxiliaires. Ainsi la résolution d'un problème comme celui des quatre cubes (instant insanity) est fortement guidée par la vision immédiate que nous avons d'un cube et notre expérience de manipulation des objets dans l'espace.

La résolution "concrète" du problème est certainement plus simple pour un humain que sa résolution sous forme "mathématique".

Ceci est vrai pour ce problème des cubes comme pour celui des disques de Banerji et des problèmes d'architecture. L'effort demandé au programme pour ne pas faire chevaucher deux éléments (contraintes OU) est démesuré par rapport au nôtre. Le programme est d'ailleurs dans l'impossibilité de faire des tests concernant la surface des éléments puisque cette notion n'apparait pas dans sa représentation.

Il y a là,à mon sens,une difficulté fondamentale; montrer qu'il est impossible de recouvrir un échiquier 8×8 dont deux coins opposés ont été ôtés par des dominos 2×1 , demande ainsi l'intervention d'une nouvelle représentation : le coloriage de l'échiquier. Montrer qu'il est de même impossible de remplir un cube 3×3×3 par des briques 2×1×1/2 , problème moins connu , nous demande encore un effort et un raisonnement différents.

H. Siklossy a abordé ces problèmes dans son programme LAWALY (1972).

Il est incontestable que le langage ALICE n'est pas encore assez riche pour formuler élégamment et simplement tous les problèmes combinatoires et que des représentations autres que celles utilisées par le programme sont à trouver.

Cette difficulté réelle liée au problème de la reconnaissance et de la manipulation des formes de l'univers concret est rencontrée par tout informaticien. Une interaction étroite entre capteur d'informations et programme est indispensable et sans elle il n'est tout simplement pas possible de dire que les programmes résolvent les problèmes que nous résolvons.

En outre, ces problèmes mettent en évidence le fait que le but du chercheur opérationnel ne doit pas être d'optimiser tout et n'importe quoi ; ces cas concrets d'organisation, par leur grande taille et leurs aspects multiples, demandent impérativement pour être résolus en un temps convenable l'utilisation systématique d'heuristiques. Les heuristiques interviennent d'ailleurs à différents niveaux:

Utilisées couramment en recherche opérationnelle pour obtenir de bonnes solutions approchées (Lin [1967] , Reiter et Scherman [1963] , Welsh et Powell [1967]) les heuristiques sont souvent et faussement confondues avec "méthodes inexactes".

En fait leur caractéristique principale consiste à arbitrer plus intelligemment que le hasard , chaque fois que la théorie laisse une possibilité de choix. A ce titre elles ne sont contradictoires ni avec la notion d'exactitude ni avec celle d'optimalité de la solution. Ainsi elles peuvent être un guide à l'intérieur d'un pas d'un algorithme polynômial et améliorer le temps d'exécution. Elles donnent une fonction d'évaluation inférieure par le choix d'une bonne relaxation, une fonction supérieure par l'obtention de solutions approchées (dans le cas d'une minimisation). Elles peuvent servir à décomposer un problème complexe, à élaborer un plan et à exécuter finement chaque étape de ce plan.

L'ordinateur serait le plus merveilleux des outils s'il ne fallait pas écrire et mettre au point les programmes.

Les langages de programmation actuels manquent de puissance et de concision (FORTRAN, ALGOL, PL1) , ou bien d'efficacité (LISP, SNOBOL, APL).

L'engouement des mathématiciens pour un langage comme APL , où de nombreuses fonctions sont préprogrammées et dans lequel les notations et objets usuels en mathématiques peuvent être utilisés, montre bien l'intérêt de rapprocher la machine de l'utilisateur; il reste cependant encore dans un tel langage à écrire les procédures.

Un langage de problèmes comme ALICE , permet, après une mise au point unique d'énoncer et de résoudre toute une famille de problèmes. Il ne reste à l'utilisateur qu'à poser formellement son problème,ce qui est bien le minimum de ce que l'on peut demander à quelqu'un qui veut en obtenir la résolution.

La généralité du programme n'entraine pas nécessairement une baisse des performances. Il faut d'ailleurs noter à ce sujet, que pour tout problème NP complet, un programme spécifique ne peut gagner qu'un facteur polynômial sans changer la nature exponentielle du temps de calcul.

L'utilisation d'une double représentation, graphique et algébrique, l'intervention de plusieurs heuristiques pour un même problème, l'adaptation du programme qui sait tirer parti des caractéristiques numériques des énoncés ont permis de meilleures performances que celles de programmes spécifiques récents.

Il est d'ailleurs remarquable que si peu d'heuristiques - cinq types de choix différents et une douzaine de critères - permettent d'aborder de façon satisfaisante d'aussi nombreux problèmes d'analyse combinatoire et de recherche opérationnelle, et ceci jusqu'à fournir, dans bien des cas, la solution optimale sans aucun retour en arrière.

De nombreuses recherches concernent actuellement la preuve de l'exactitude ou la génération automatique de programmes. Dans ce domaine on est encore loin de pouvoir obtenir en machine des programmes pour, par exemple, chacun des six premiers exercices proposés au chapitre I . Or, on a vu qu'en prenant un cadre général et en utilisant un système de représentation commun, il était possible de résoudre ces problèmes par un programme unique ; programme qu'on est ensuite libre de simplifier et d'optimiser automatiquement pour traiter un cas spécifique. En outre, si le plus souvent, le programme résout les problèmes par interprétation , il est capable de plus que cela : le système de manipulation permet notamment d'engendrer de nouvelles contraintes et d'effectuer ainsi une étude non triviale sur l'énoncé a priori.

BIBLIOGRAPHIE

Les références de cette bibliographie ont constitué le support constant du travail précédemment présenté. Tous les titres n'ont pas été cités dans le texte, seuls y ont en effet figurés ceux qui présentaient un caractère de généralité ou de nouveauté. Une telle réunion d'ouvrages - articles de présentation globale, rapports techniques sur un énoncé particulier, recueils de problèmes et de données numériques - n'existait pas sur le sujet abordé. Elle pourra, je l'espère, servir de nouveau à un lecteur désireux de travailler dans le même domaine.

ABADIE. J. ed. (1967), Nonlinear Programming, John Wiley & Sons.

——— ed. (1970), Integer and Nonlinear Programming, American Elsevier.

ADAM A . (1973)"Gadget, un programme de génération automatique sur les graphes et des ensembles".Thèse 3ème cycle.Paris.

AGARD J., ARABEYRE J.P., VAUTIER J. (1967)"Génération automatique des rotations d'équipages", R.I.R.O. n°6.

AGIN N. (1966) "Optimum Seeking with Branch and Bound", Management Science, 13, 176-185.

ALBRECHT R. (1968)"Bestimmung minimaler Wege in endlichen, gerichteten, bewerteten Graphen". Computing 3, 184-193.

AMAREL S.(1970) "On representations of problems of reasoning about action". Machine Intelligence 3 p 131-172. Michie Editeur. Edinburgh.

APPELGREN L.H. (1971) "Integer Programming Methods for a Vessel Scheduling Problem", Transportation Science, 5, 64-78.

ANDERSON I. (1971) "Perfect Matchings in a Graph", J. Comb. Theory 10, 183-186.

ARABEYRE J.P., FEARNLEY J., STEIGER F.C. and TEATHER W. (1969), "The Airline Crew Scheduling Problem : A Survey", Transportation Science 3, 140-163.

ARMOUR G.C., et BUFFA E.S. (1965), "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities", Man.Sci. 9, 294-309.

ARONOFSKY J., ed. (1969), Progress in Operations Research, Vol. 3 : Relationship between Operations Research and the Computer, John Wiley & Sons;

ARSAC J. (1968) La Science informatique.Dunod,Paris.

ARTHANARI T.S. et RAMAMURTHY K.G. (1970) " A Branch-and-Bound Algorithm for Sequencing n-Jobs on m-Parallel Processors", Opns. Res 7 , 147-156.

ASHOUR S. (1970a) "A Branch-and-Bound Algorithm for Flow Shop Scheduling problems", AIIE Trans. II, 172-176.

——— (1970b) "An Experimental Investigation and Comparative Evaluation of Flow Shop Scheduling Techniques", Opns. Res. 18, 541-548.

ASHOUR S. et CHAR A.R. (1971) "Computational Experience on 0-1 Programming Approaches to Various Combinatorial Problems", Journal of the Operations Research Society of Japan , 13, 78-107.

BALAS E. (1964) "Extension de l'algorithme additif à la programmation en nombres entiers et à la programmation non linéaire", Comptes Rendus hebdomadaires de l'Académie des Sciences de Paris, 258, 5136-5139.

——— (1965) "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", Opns. Res. 13, 517-546.

——— (1967) "Discrete Programming by the Filter Method", Opns. Res. 15, 915-957.

——— (1968) "A Note on the Branch-and-Bound Principle", Opns. Res. 16, 442-445; errata:Opns. Res. 16,886.

BALAS E. et MIHOC G. (1965)"The problem of optimal timetables" Revue Roumaine Maths Pures et Appliquées 10.

BALAS E. et PADBERG M.W. (1970) "On the set covering problem", Man. Sci. Res.Rep. No. 197, Carnegie-Mellon University.

BALAS E. et PADBERG M.W. (1974)"On the set covering problem II" Opns. Res. 25, 520-527.

BALINSKI M.L. (1961) "Fixed Cost Transportation Problems", Nav. Res.Log. Quart.11, 41-54.

——— (1964) "On Finding Integer Solutions to Linear Programs", IBM Data Processing Division, 225-248.

——— (1965) "Integer Programming : Methods, Uses, Computation", Man. Sci. 12, 253-313.

——— (1967) "Some General Methods in Integer Programming", 220-247, in J. ABADIE (editor), Integer and Non linear Programming.

——— (1969) "Labelling to Obtain a Maximum Matching", Combinatorial Mathematics and its Applications, R.C. Bose and T.A. Dowling (eds.), University of North Carolina Press, 585-602.

——— (1970a)"On Recent Developments in Integer Programming", 267-302, in Kuhn.

——— (1970b) "On Maximum Matching, Minimum Covering and their Connections", 303-312, in Kuhn.

BALINSKI M.L. et QUANDT R.E. (1964) "On an Integer Program for a Delivery Problem", Opns.Res. 12, 300-304.

BALINSKI M.L. et SPIELBERG K. (1969) "Methods for Integer Programming : Algebraïc, combinatorial and enumerative", 195-292 in ARONOFSKY, J., editor, Progress in Operations Research, Vol.III : Relationship between Operations Research and the Computer, John Wiley and Sons, New York.

BEALE E.M.L. (1970) "Selecting an Optimum Subset", pp. 451-462 in J. ABADIE (editor), Integer and Nonlinear Programming

BEALE E.M.I. et TOMLIN J.A. (1969) "Special Facilities in a General Mathematical Programming System for Non-convex Problems Using Ordered Sets of Variables" 8.21-8.28 in LAWRENCE J. editor, Proceedings of the Fifth International Conference of Operational Research, Venice, Tavestock Publ. London.

BELLMAN R.E. et DREYFUS S.E. (1962) Applied Dynamic Programming, Princeton University Press.

BELLMORE M. et MALONE J.C. (1971) "Pathology of Traveling-Salesman Subtour Elimination Algorithms", Opns.Res. 19, 278-307.

BENDERS J.F. (1962) "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik 4, 238-252.

BERGE C. (1960) "Les problèmes de coloration en théorie des graphes". I.S.U.P. vol. IX,2, Paris.

——— (1970) Graphes et Hypergraphes. Dunod.Paris.

BERTIER P. (1966) "Procédure pour élaborer des circuits de distribution." Metra n°8.

BERTIER et ROY B. (1964) "Procédure de résolution pour une classe de problèmes pouvant avoir un caractère combinatoire", Cahiers du Centre d'Etudes de Recherche Opérationnelle, 6, 202-208.

BESSIERE F. (1968) "Sur la recherche du nombre chromatique d'un graphe par un programme linéaire en nombres entiers", Revue Française d'Informatique et de Recherche Opérationnelle, 9, 143-148.

BITNER R. et REINGOLD E. "Backtrack programming techniques". CACM Vol. 18 n°11 p 651-656.

BOBROW D. "Natural Language Input for a Computer Problem Solving System", Doctoral Dissertation, Mathematics Dept., Massachusetts Institute of Technology, Cambridge, Mass.

BOOT C.J.T. et THEIL H. (1964) "A Procedure for Integer Maximization of a Definite Quadratic Function", pp. 667-682 in KREWERAS G. and MORLAT G., editors, Proceedings of the Third International Conference on Operations Research, Dunod, Paris and English Universities Press.

BOUVIER B. et MESSOUMIAN G. (1965) "Programmes linéaires en variables bivalentes (Algorithme de Balas)". Thèse, Faculté des Sciences, Université de Grenoble.

BOWMAN E.H. (1969) "The Schedule-Sequencing Problem", Opns.Res. 7, 621-624.

——— (1960) "Assembly Line Balancing by Linear Programming", Opns.Res.8. 385-389.

BRADLEY G.H. (1969) "Equivalent Integer Programs", 455-463, in Lawrence.

——— (1971a) "Heuristic Solution Methods and Transformed Integer Linear Programming Problems", TEP. No. 43, Dept. of Administrative Science Yale University.

——— (1971b) "Transformation of Integer Programs to Knapsack Problems", Discrete Math 1, 29-45.

BURSTALL R.M. (1965) "A Tree Searching Method for Solving Integer Linear Inequalities", Experimental Programming Report No. 10, University of Edinburgh.

——— (1966) "A Heuristic Method for a Job-Scheduling Problem" Operational Research Quarterly, 17, 291-304.

BUSAKER R.G. et SAATY T.L. (1965) Finite Graphs and Networks : An Introduction with Applications, Mc Graw-Hill.

BYRNE J.L. et PROLL L.G. (1969) "Initialising Geoffrion's Implicit Enumeration Algorithm for the Zero-One Linear Programming Problem", Computer Journal , 381-384.

CABOT V.A. et HURTER A.P. (1968) "An Approach to 0-1 Integer Programming", Opns. Res. 16, 1206-1211.

CAMION P. (1960) "Une méthode de résolution par l'algèbre de Boole des problèmes combinatoires où interviennent des entiers", Cahiers du Centre d'Etudes de Recherche Opérationnelle, 2, 234-289.

CARLIER J. (1975) "Disjonctions dans les ordonnancements" RAIRO, Vol 2.

——— Thèse de troisième cycle, Paris VI.

CARLSON R.G. et NEMHAUSER G.L. (1966) "Scheduling to Minimize Interaction Cost", Opns. Res.14, 52-58.

CARROLL L. (1966) Logique sans peine. Hermann, Paris.

CHARLTON J.M. et DEATH C.C. (1970) "Generalized Machine Scheduling Algorithm", Opnal Res. Quart.21, 127-134.

CHARNES A. et COOPER W.W. (1961) Management Models and In dustrial Applications of Linear Programming, Vols.1 and 2, John Wiley & Sons.

CHATY G. CHEIN M. MARTIN P. PETOLLA G. (1975) "Register allocation invariants in dags" 6th CGTC Florida Atlantic University.

CLARKE G. et WRIGHT S.W. (1964) "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", Opns. Res. 12, 568-581.

COBHAM A. FRIDSHAL R. et NORTH J.H. (1961) "An Application of Linear Programming to the Minimization of Boolean Functions", Res. Rep. RC-472,IBM.

——— (1962) "A Statistical Study of the Minimization of Boolean Functions Using Integer Programming", Res. Rep. RC-756, IBM.

COFFMAN, SETHI,BRUNO,ULLMAN,KOHLER,STEIGLITZ,GRAHAM (1976): "Computer job shop scheduling problems" à paraître.

COOK S.A. (1971) "The complexity of theorem-proving procedures" 3 STOC., 151-158

COOPER L. et DREBES C. (1967) "An Approximate Solution Method for the Fixed Charge Problem", Nac. Res. Log. Quart. 14, 101-113.

COURTILLOT M. (1965) "Sur la résolution des programmes à solutions entières", Revue Française de Traitement de l'Information-Chiffres, 8, 81-94.

CHRISTOFIDES N. (1971) An algorithm for the chromatic number of a graph. Computer Journal n°14-1

——— (1973) Large scheduling problems with bivalent costs. Computer Journal n°16-3.

CHRISTOFIDES N. et EILON S. (1969) "An Algorithm for the Vehicle Dispatching Problem", Opnal. Res. Quart. 20, 309-318.

CORNEIL D.G. et GRAHAM B. (1973)"An algorithm for determining the chromatic number of a graph" Siam j. Comput. Vol 2 n°4.

CROWSTON W.B. et PIERCE J.F. (1971) "Tree Search Algorithms in Quadratic Assignment Problems", Naval Research Logistics Quarterly, 18, 1-36.

DAKIN J. (1965) "A Tree-Search Algorithm for Mixed-integer Programming Problems", Computer Journal, 8, 250-255.

DANTZIG G.B. (1957) "Discrete Variable Extremum Problems", Opns. Res. 5, 266-277.

——— (1959) "Notes on Sol ing Linear Programs in Integers", Nav. Res. Log. Quart. 6, 75-76.

——— (1963) Linear Programming and Extension. University Press.

DANTZIG G.B., FULKERSON D.R. et JOHSON S.M. (1954) "Solution of a Large Scale Travelling Salesman Problem", Opns. Res. 2, 393-410.

——— (1959) "On a Linear Programming, Combinatorial Approach to the Travelling Salesman Problem", Opns. Res.7, 58-66.

DANTZIG G.B. et RAMSER J.H. (1959) "The Truck Dispatching Problem", Man. Sci.6, 80-91.

DANTZIG G.B. et VEINOTT A.F., Jr., eds. (1968), Mathematics of the Decision Sciences, Part I, Amer. Math. Soc.

DAVID A. et ROUCAIROL C. (1973) : Un algorithme d'aide à la conception en architecture. RAIRO V.3. p.69-81

DAVIS E.W. et HEIDORN G.E. (1971) "An Algorithm for Optimal Project Scheduling under Multiple Ressource Constraints", Man. Sci.17, B803-B816.

DAVIS P.S. et RAY T.L. (1969) "A Branch-Bound Algorithm for the Capacitated Facilities Location Problem", Nav. Res. Log. Quart.16, 331-344.

DESLER J.F. et HAKIMI S.L. (1969) "A Graph-Theoretic Approach to a Class of Integer Programming Problems", Opns. Res.17, 1017-1033.

DE WERRA D. (1969) Thèse de l'Ecole Polytechnique fédérale de Lausanne.

——— (1970) "On some Combinational Problems Arising in Scheduling", CORS 8, 165-175.

DRAGAN I. (1968) "Un Algorithme Lexicographique pour la Résolution des Programmes Polynomiaux en Variables Entières", RIRO 2, 81-90.

——— (1969) "Un Algorithme Lexicographique pour la Résolution des Programmes Lineaires", Man. Sci.16, 246-252.

——— (1970) "An Improvement of the Lexicographical Algorithm for Solving Linear Discrete Programming Problems", CORE Discussion Paper No. 7029, Louvain, Belgium.

DIJKSTRA E.W. (1959) "A Note on Two Problems in Connexion with Graphs", Numer. Mathematik 1, 269-271.

DRYSDALE J.K., et SANDIFORD P.J. (1969) "Heuristic Warehouse Location A Case History Using a New Method", CORS 7, 45-61.

EASTMAN W.L. (1958) "Linear Programming with Pattern Constraints", Ph.D. Dissertation , Harvard University.

ECHOLS R.E., et COOPER L. (1968) "Solution of Integer Linear Programming Problems by Direct Search", JACM 15, 75-84.

EDMONDS J. (1962) "Covers and Packings in a Family of Sets", Bull. Ann. Math. Soc. 68, 494-499.

——— (1965a) "Paths, Trees and Flowers", Can. J.Math 17, 449-467.

——— (1965b) "Maximum Matching and a Polyhedron with 0,1-Vertices", J. Res. Nat. Bur. Stds. 69B, 125-130.

——— (1967) "An Introduction to Matching", Lecture Notes, University of Michigan Summer Eng. Conf.

EDMONDS J., et FULKERSON D.R. (1970) "Bottleneck Extrema", J. Comb. Theory 8, 299-306.

EDMONDS J., et JOHNSON E.L. (1970) "Matching : A Well-Solved Class of Integer Linear Programs", Proc. of the Calgary Int. Conf. on Comb. Structures and their Appl., 89-92, Gordon and Breach.

EFROYMSON M.A., et RAY T.L. (1966) "A Branch-and-Bound Algorithm for Plant Location", Opns. Res. 14, 361-368.

EILON D.M. (1966) "Building up a Sequence of Optimum Media Schedules", Operational Research Quarterly, 17, 413-424.

ELMAGHRABY S.E., et WIG M.K. (1970) "On the Treatment of Cutting Stock Problems as Diophantine Programs", Dept. of Industrial Engineering, North Carolina State University.

ELSHAFEI A.M. (1972) "Facilities Location : Formulations, Methods of Solution, Applications and Some Computational Experience", Memo n°276, The Institute of National Planning, Nasr City, Cairo.

ERDOS P. et HAJNAL (1966) On chromatic number of graphes and set systems Acta Math. Acad. Soc. Hung. 17.

FAALAND B. (1970) "Generalized Equivalent Integer Programs and Canonical Problems", Tech. Rep. No. 21, Dept. of Operations Research, Stanford University.

——— (1971) "Solution of the Value Independent Knapsack Problem by Partitioning", Tech. Rep. No. 22, Dept. of Operations Research, Stanford University.

FAURE R. et MALGRANGE Y. (1965)"Nouvelle recherche sur la résolution des programmes lineaires en nombres entiers." Revue Gestion, Dunod.

FAURE R. (1968) Elements de Recherche Opérationnelle.Gauthiers Villars.

FAYARD D. et PLATEAU G. (1972) "Résolution du problème du knapsack en variables bivalentes. Comparaison de méthodes", Bulletin de la Direction des études et Recherches E.D.F., Série C, Mathématiques, Informatique, 1, 5-42.

FEIGENBAUM E. (1968)"Artificial intelligence : themes of the second decade." Stanford Art. Int. Project. Californie. Memo n° 67, 15.

FIKES R. (1968) "A Heuristic Program for Solving Problems Stated as Nondeterministic Procedures", Doctoral Dissertation, Computer Science Department, Carnegie-Mellon University.

——— (1968) "Stating Problems as Procedures to a General Problem Solving Program", Proceedings of the Fourth Systems Symposium, Case Western Reserve University, Cleveland, Ohio.

FLEISCHMANN B. (1967) "Computational Experience with the Algorithm of Balas", Operations Research, 15, 153-155.

FLORIAN M., TREPANT P., and McMAHON G.B. (1971) "An Implicit Enumeration Algorithm for the Machine Sequencing Problem", Man. Sci. 17, B782-B792.

FLOYD R. (1966) "Assigning Meaning to Programs", Proceedings of the American Mathematical Society's Symposium on Applied Mathematics, Vol. 19.

FLOYD R. (1967) "Nondeterministic Algorithms", Journal of the Association for Computing Machinery, Vol. 14, No 4, pp. 636-644.

FOATA D. et SCHUTZENBERGER M.P. (1970) "Théorie géométrique des polynômes eulériens". Lecture notes in Mathematics n° 138. Springer Verlag.

FORCADE R. (1973) "Smallest maximal-matching in the graph of the d dimensional cube." J. of Combinatorial Theory (B) 14 p 153-156.

FORD L.R., Jr., et FULKERSON D.R. (1962) Flows in Networks, Princeton University Press.

FULKERSON, NEMHAUSER et TROTTER (1974) Mathematical Programming Study 2.

FORTET R. (1959) "L'algèbre de Boole et ses applications en recherche opérationnelle", Cahiers du Centre d'Etudes de Recherche Opérationnelle, 1, 5-36.

——— (1960) "Applications de l'algèbre de Boole en recherche opérationnelle", Revue Française de Recherche Opérationnelle, 4, 17-26.

GARFINKEL R.S. (1971) "An Improved Algorithm for the Bottleneck Assignment Problem", Opns. Res. 19, 1747-1751.

——— (1972) "On Partitioning the Feasible Set in a Branch-and-Bound Algorithm for the Asymmetric Traveling Salesman Problem", Opns. Res.

GARFINKEL R.S. et NEMHAUSER G.L. (1969) "The Set Partitioning Problem : Set Covering with Equality Constraints", Operations Research, 17, 848-856.

——— (1970) "Optimal Political Districting by Implicit Enumeration Techniques", Management Science, 16, 495-508.

——— (1972a) 164-183 in GEOFFRION A., editor, Perspectives on Optimization : A Collection of Expository Articles, Addison-Wesley.

——— (1972b) Integer Programming, Wiley, New York.

GATELY D. (1970) "Investment Planning for the Electric Power Industry : An Integer Programming Approach", Res. Rep. 70, Dept. of Economics, University of West Ontario, London, Canada.

GAVET J.W., et PLYTER N.V. (1966) "The Optimal Assignment of Facilities to Locations by Branch and Bound", Opns. Res. 14, 210-232.

GENUYS F. (1962) Application de la programmation linéaire en nombres entiers à un problème de découpe. Proc. IFIP Conférence.

GEOFFRION A.M. (1967) "Integer Programming by Implicit Enumeration and Balas' Method", SIAM Rev. 7, 178-190.

——— (1969) "An Improved Implicit Enumeration Approach for Integer Programming", Opns. Res. 17, 437-454.

——— ed. (1972) Perspectives on Optimization : A Collection of Expository Articles, Addison-Wesley.

GEOFFRION A.M. et MARSTEN R.E. (1972) "Integer Programming Algorithms : A Framework and State-of-the-art Survey", Management Science, 18, 465-491.

GHARE P.M., MONTGOMERY D.C. et TURNER W.C. (1971) "Optimum Interdiction Policy for a Flow Network", Nav. Res. Log. Quart 18, 37-46.

GHOUILA-HOURI A. (1964)"Flots et tensions dans un graphe."Ann. Sc. ec. Norm. sup 3ème serie 81, p. 268.

GIGLIO R.J., et WAGNER G.M. (1964) "Approximate Solutions to the Three-Machine Scheduling Problem", Opns. Res. 12, 306-324.

GILMORE P.C., et GOMORY R.E. (1961) "A Linear Programming Approach to the Cutting Stock Problem", Opns. Res. 9, 849-859.

——— (1963) "A Linear Programming Approach to the Cutting Stock Problem, Part II", Opns. Res. 11, 863-888.

——— (1965) "Multistage Cutting Stock Problems of Two and More Dimensions", Opns. Res. 13, 94-120.

——— (1966) "The Theory and Computation of Knapsack Functions", Opns. Res. 14, 1045-1074.

GINSBURGH V. et VAN PEETERSEN A. (1969) "Un algorithme de programmation quadratique en variables binaires", Revue Française d'Informatique et de Recherche Opérationnelle, 3, V2, 57-74.

GLOVER F. (1965) "A Bound Escalation Method for the Solution of Integer Linear Programs", Cahiers Cent. d'Etudes Recherche Opérationnelle 6, 131-168.

——— (1968a) "Surrogate Constraints", Opns. Res. 16, 741-749.

——— (1968b) "A Note on Linear Programming and Integer Feasibility", Opns. Res. 16, 1212-1216.

GLOVER F. et WOOLSEY R.E.D. (1972a) "Further Reduction of Zero-One Polynomial Programming Problems to Zero-One Linear Programming Problems", Operations Research, 21, 156-161.

——— ——— (1972b) "Note on Converting the 0-1 Polynomial Programming Problem to a 0-1 Linear Program", Management Science Report Series n° 72-3, University of Colorado, Boulder, Colorado.

GLOVER F. et ZIONTS S. (1965) "A Note on the Additive Algoritm of Balas", Opns. Res. 13, 546-549.

GOLOMB S.W. et BAUMERT L.D. (1965) "Backtrack Programming", Journal of the Association for Computing Machinery, 12, 516-524.

GOMORY R.E. (1958) "Outline of an Algorithm for Integer Solutions to Linear Programs", Bull. Amer. Math. Soc. 64, 275-278.

——— (1960) "Solving Linear Programming Problems in Integers", 211-216, in Bellman and Hall.

——— (1963) "An Algorithm for Integer Solutions to Linear Programs", 269-302, in Graves and Wolfe. (Originally appeared as Princeton-IBM Math. Res. Project Tech. Rep. No. 1, 1958).

GONDRAN M. (1970) "Programmation entière non linéaire", Revue Française d'Informatique et de Recherche Opérationnelle, 4, R-3, 107-110.

——— (1972) "Programmes en nombres entiers sans contraintes : génération des optimums locaux dans le cas quadratique", Revue Française d'Informatique et de Recherche Opérationnelle, 6, R-2, 80-83.

GONDRAN M. (1973) Un algorithme de coupes efficace par la méthode des congruences décroissantes, note EDF, HI 1234/02.

——— (1974) Problèmes combinatoires et programmation en nombres entiers. Thèse Etat. Paris.

GONDRAN et LAURIERE J.L. (1974) Un algorithme pour le problème de partitionnement, R.A.I.R.O., n° V-1, p. 25-38.

——— ——— (1975) Un algorithme pour les problèmes de recouvrement, R.A.I.R.O. n° V-2, p. 33-51.

GOTLIEB C.C. (1962) "The construction of class teacher timetables" Processing of IFIP Congress, 91-98

GRANOT F. et HAMMER P.L. (1970) "On the use of Boolean Functions in 0-1 Programming", Report n° 70, Faculty of Industrial and Management Engineering, Technion, Israel.

GRAVES G.W. et WHINSTON A.B. (1968) "A New Approach to Discrete Mathematical Programming", Man. Sci. 15, 177-190.

GRAVES R.L. et WOLFE P. (1963) ; Recent advances in Mathematical Programming. Mc Graw Hill.

GREEN C. (1969) "Application of Theorem Proving to Problem Solving", Proceedings of the International Joint Conference on Artificial Intelligence, Washington, D.C.

GROSS O. (1958) "The Bottleneck Assignment Problem", RAND, P-1630.

GUIGNARD M. (1971) "Une condition d'optimalité en programmation en nombres entiers", Revue Française d'Informatique et de Recherche Opérationnelle, 5, R-2, 108-113.

GUIGNARD M. et SPIELBERG K. (1970) "The State Enumeration Method for Mixed Zero-One Programming" Mathematical Programming Symposium, The Hague, The Netherlands.

GUIGNARD M. et SPIELBERG K. (1973) "A Realization of the State Enumeration Procedure", IBM Philadelphia Scientific Center, Technical Report n° 320-3025.

GUPTA J. et WALVEKAR A. (1969) "Sequencing n Jobs on m Pa allel Processors", Opns. Res. 6, 295-298.

HA DLEY G. (1962) Linear Programming, Addison-Wesley.

——— (1964) Nonlinear and Dynamic Programming, Addison-Wesley.

HAKIMI S.L. (1965) "Optimum Distribution of Switching Centers in a Communication Network and some Related Graph Theoretic Problems", Operations Research, 13, 462-475.

HAKIMI S.L. et MAHESHWARI S.V. (1972) "Optimum Location of Centers in Networks", Operations Research, 20, 967-973.

HALDI J. (1964) "25 Integer Programming Test Problems", Working Paper No. No. 43, Graduate School of Business, Stanford University.

HALDI J. et ISAACSON L.M. (1965) "A Computer Code for Integer Solutions to Linear Problems", Opns. Res. 13, 946-959.

HALL P. (1935)"On Representatives of Subsets." J. London Math. Soc. 10, 26-30.

HAMMER P.L. (1968) "Plant Location - a Pseudo-Boolean Approach", Israel Journal of Technology, 6, 330-332.

——— (1969a) "Increasing the Capacity of a Network", CORS 7, 83-91.

——— (1969b) "Time Minimizing Transportation Problems", Nav. Res. Log. Quart. 16, 345-357.

——— (1971a) "A B-B-B Method for Linear and Nonlinear Bivalent Programming", in Avi-Itzhak.

——— (1971b) "BABO-A Boolean Approach for Bivalent Optimization", Centre des Recherches Mathématiques, University of Montreal.

——— (1974)"Boolean procedures for bivalent programming", in Mathematical programming theory and applications, North Holland.

HAMMER P.L. et HANSEN P. (1972) "Quadratic 0-1 Programming", CORE Discussion Paper n° 7219.

HAMMER P.L. et NGUYEN S. (1972) "APPOSS- A Partial Order in the Solution Space of Bivalent Programs", Université de Montréal, Centre de Recherches Mathématiques, CRM-163.

HAMMER P.L. et PELED U.N. (1972) "On the Maximization of a Pseudo- Boolean Function", J.A.C.M. , Vol.19 n°2, 265-282.

HAMMER P.L. et RUBIN A.A. (1970) "Some Remarks on Quadratic Programming with 0-1 Variables", Revue Française d'Informatique et de Recherche Opérationnelle, 4, 67-79.

HAMMER P.L. et RUDEANU S. (1968) Boolean Methods in Operations Research and Related Areas, Springer-Verlag, Berlin .

——— (1969) "Pseudo-Boolean Programming", Opns. Res. 17, 233-261.

HANSEN P. (1970) "A Branch and Bound Algorithm for Nonlinear 0-1 Programming", University of Brussels, Belgium.

——— (1971) "Quadratic 0-1 Programming by Implicit Enumeration", University of Brussels, Belgium.

——— (1974) "Programmation mathématique en variables 0-1. Thèse. Bruxelles.

HANSEN P. et KAUFMAN L. (1972) "Comparaison d'algorithmes pour le problème de la localisation des entrepôts", pp. 281-294 dans J. BRENNAN (editor) Operational Research in Industrial Systems, English Universities Press, London.

HARARY F.(1969) Graph Theory, Addison-Wesley.

HEALEY W.C., Jr (1964) "Multiple Choice Programming", Opns. Res. 12, 122-138.

HELD M. et KARP R.M. (1970) "The Traveling-Salesman Problem and Minimum Spanning Trees", Opns. Res. 18, 1138-1162.

——— (1971) "The Traveling-Salesman Problem and Minimum Spanning Trees : Part II", Math. Prog. 1, 6-25.

HELLER G. (1957) "On Linear Systems with Integral Valued Solution", Pac. J. Math. 7, 1351-1364.

HENRY G. (1968) "Recherche d'un réseau de dépôts optimum", Revue Française d'Informatique et de Recherche Opérationnelle, 2, 69-80.

HERZ J.C. (1970) Exposé et principe de résolution des problèmes d'emploi du temps scolaires. Informatique et Sciences Humaines n° 5.

HEURGON E. (1967) Programmation linéaire en nombres entiers. Thèse 3ème cycle, U. Paris.

——— (1972) Un problème de recouvrement : l'habillage des horaires des lignes d'autobus, R.A.I.R.O. 6ème année, n° V-1, p. 13-29.

HILLIER P.S. (1969a) "Efficient Heuristic Procedures for Integer Linear Programming with an Interior", Opns. Res. 17, 600-637.

——— (1969b) "A Bound-and-Scan Algorithm for Pure Integer Linear Programming with General Variables", Opns. Res. 17, 638-679.

HILLIER F.S. et CONNORS M.M. (1967) "Quadratic Assignment Problem Algorithms and the Location of Indivisible Facilities", Man. Sci. 13, 42-57.

HIRSH W.M. et DANTZIG G.B. (1968) "The Fixed Charge Problem", Nav. Res. Log. Quart. 15, 413-424.

HOUSE R.W., NELSON L.D. et RADO J. (1966) "Computer Studies of a Certain Class of Linear Integer Problems", 241-280, in Lavi and Vogl).

HUARD P. (1967) "Resolutions of Mathematical Programming with Nonlinear Constraints by the Method of Centers", 207-219, in Abadie.

——— (1970) "Programmes Mathématiques Nonlineaires à Variables Bivalentes", 313-322, in Kuhn.

HUARD P. et LUDOT (1970) "Contribution à l'estimation de la sécurité des réseaux" Revue EDF n°1, 69-100.

HU T.C. (1963) "Multi-Commodity Nework Flows", Opns. Res. 11, 344-360.

——— (1969) Integer Programming and Network Flows, Addison-Wesley.

——— (1970) "On the Asymptotic Integer Algorithm", Lin. Alg. and Appl. 3, 279-294.

——— (1971) "Some Problems in Discrete Optimization", Math. Prog. 1, 102-112.

IBARAKI T. LIU T.K., BAUGH C.R. et MUROGA S. (1969) "Implicit Enumeration Program for Zero-One Integer Programming", Department of Computer Science, University of Illinois.

IGNALL E. et SCHRAGE L. (1965) "Applications of the Branch-and-Bound Technique to Some Flow-Shop Scheduling Problems", Opns. Res. 13, 400-412.

JAMES W. (1966) "Some Remards on a Numerical Example of Balas and on the note by Glover and Zionts", Operations Research, 14, 935-941.

JARVINEN P., RAJALA J. et SINERVO M. (1972) "A Branch-and-Bound Algorithm for Seeking the p-Median", Operations Research, 20, 173-178.

JENSEN P.A. (1971) "Optimum Network Partitioning", Opns. Res. 19, 916-932.

JEROSLOW R.G. (1972) "There Cannot Be Any Algorithm for Integer Programming with Quadratic Constraints", Operations Research, 20, 228-234.

JEROSLOW R.G. et KORTANEK K.O. (1969) "Dense Sets of Two Variable Integer Programs Requiring Arbitrarily Many Cuts by Fractional Algorithms", Man. Sci. Res. Tep. No. 174, Carnegie-Mellon University.

——— (1971) "On an Algorithm of Gomory", SIAM J. 21, 55-60.

JOHNSON D.S. (1973) "Approximation algorithms for combinatorial problems" Proc. 5th ACM Symposium of computing.

JOHSON S. (1964) "Optimal two and three stage production schedules with set up times included." Naval Res. Log. Quaterly vol. 1 n°1, p. 61.

JONES A.P. et SOLAND R.M. (1969) "A Branch-and-Bound Algorithm for Multi-level Fixed Charge Problems", Man. Sci. 16, 67-76.

KALIMON B.A. (1971) "Note Regarding A New Approach to Discrete Mathematical Programming", Man. Sci. 17, 777-778.

KAPLAN S. (1966) "Solution of the Lorie-Savage Problem and Similar Integer Programming Problems", Opns. Res. 14, 1130-1136.

KARP R.M. (1972) "Reducibility Among Combinatorial Problems", Tech. Rep. 3, Computer Science, University of California, Berkeley.

KAUFMAN L. (1971) "La localisation des entrepôts", Revue Belge de Statistique et de Recherche Opérationnelle, 11, 22-34.

KELLEY (1961) "Critical Path Planning and Scheduling : Mathematical Basis." Operations Res. 9, 296-320.

KHUMAWALA B.M. (1972) "An Efficient Branch-and-Bound Algorithm for the the Warehouse Location Problem", Management Science, 18, 718-731.

KIANFAR F. (1971) "Stronger Inequalities for 0,1 Integer Programming Using Knapsack Functions", Opns. Res. 19, 1374-1392.

KIRCHASSNER K. (1963)"Uber Zuteilungs probleme einer Universitat" Jahrestag. d. DGU, Brauschweig.

KIRBY M.J.L. et SCOBEY P.F. (1970) "Production Scheduling on N Identical Machines", CORS 8, 14,27.

KOLESAR P.J. (1967) "A Branch and Bound Algorithm for the Knapsac Problem", Man. Sci. 13, 723-735.

KOLNER T.N. (1966) "Some Highlights of a Scheduling Matrix Generator System", United Airlines.

KOOPMANS T.C. et BECKMANN M.J. (1957) "Assignment Problems and the Location of Economic Activities", Econometrica 25, 53-76.

KORTE B., KRELLE W. et OBERHOFER W. (1969) "Ein Lexicographischer Suchalgorithmus zur Lösung allgemeiner ganzzahliger Programmierungsaufgaben", Unternehmensforschung, 13, 73-98 and 13, 171-192.

——————————— (1970) "Ein Lexicographischer Suchalgotithmus zur Lösung Allgemeiner Ganzzahliger Programmie - rungsaufgaben-Nachtrag, Unternehmensforschung, 14, 228-234.

KRAITCHIK M. (1926) Le problème des reines. Bruxelles.

KRAMER F. et KRAMER H. (1969)"Un problème de coloration des sommets d'un graphe." C.R. Acad. Sc. Paris 268, A46.

KROLAK P. (1969) "Computational Results of an Integer Programming Algorithm", Opns. Res. 17, 743-749.

KROLAK P., FELTS W. et MARBLE G. (1971) "A Man Machine Approach toward Solving the Traveling Salesman Problem", CACM 14, 327-334.

KRUSKAL J.B. (1956) "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Probmem", Proc. Amer. Math. Soc. 2, 48-50.

KUEHN A.A. et HAMBURGER M.J. (1963) "A Heuristic Program for Locating Warehouses", Man. Sci. 9, 643-666.

KUHN H.W. (1955) The hungarian method for the assignement problem. Naval Research Quarterly 2.

KUHN H.W et BAUMOL W.J. (1962) "An Approximate Algorithm for the Fixed Charge Transportation Problem", Nav. Res. Log. Quart 9, 1-15.

KUNZI H.P. et OETTLI W. (1963) "Integer Quadratic Programming", 303-308. in Graves and Wolfe.

LAND A.H. et DOIG A.G. (1960) "An Automatic Method for Qolving Discrete Programming Problems", Econometrica, 28, 497-520.

LAUGHHUNN D.J. (1970) "Quadratic Binary Programming with Applications to Capital-Budgeting Problems", Opns. Res. 18, 454-461.

LAURIERE J.L.(1971)"Sur la coloration de certains hypergraphes."Thèse 3ème cycle Paris.

——— (1974) "Problèmes d'emploi du temps et algorithme de coloration des hypergraphes." C.R. Acad. Sc Paris. 278.

——— (1974) "Un programme d'élaboration des emplois du temps scolaires." Congrès aide à la décision AFCET Paris.

LAVALLEE I. (1973)"Un algorithme de détermination de couverture de cardinal minimum."RAIRO R.2.

LAS VERGNAS M. (1970)"Sur les systèmes de représentants distincts." C.R. Acad. Sc. Paris , 47-52.

LAWLER E.L. (1963) "The Quadratic Assignment Problem", Man. Sci. 9, 586-599.

——————— (1966) "Covering Problem : Duality Relations and a New Method of Solution", SIAM J. 14, 1115-1132.

LAWLER E.L. et BELL M.D. (1966) "A Method for Solving Discrete Optimization Problems", Opns. Res. 14, 1098-1112.

LAWLER E.L. et WOOD D.E. (1966) "Branch-and-Bound Methods : A Survey", Opns. Res. 14, 699-719.

LEHMER D.E. (1960) "Teaching Combinatorial Tricks to a Computer", in BELLMAN R., editor "Combinatorial Analysis", Proceedings of Symposia in Applied Mathematics, American Mathematical Society, Providence.

LE GARFF et MALGRANGE (1964) "Résolution des programmes linéaires par une méthode booléenne compacte" 795-703 dans KREWERAS G. et MORLAT G. (éditeurs), Actes de la 3ème Conférence nternationale de Recherche Opérationnelle, Dunod et English Universities Press.

LEMAIRE B. (1971)"Problèmes de tournées avec contraintes multiples". Thèse Docteur Ingénieur Paris.

LEMKE C.E. et SPIELBERG K. (1967) "Direct Search Zero-One and Mixed-Integer Programming", Operations Research, 15, 892-914.

LEMKE C.E., SALKIN H.M. et SPIELBERG K. (1971) "Set Covering by Single-Branch Enumeration with Linear-Programming Subproblems", Operations Research, 19, 998-1022.

LENSTRA K. et RINNOOY KAN A.H.G. (1975) : Some simple applications of the travelling salesman problem. Operations Research,26,717-734.

LEVY J. (1967) "An Extended Theorem for Location on a Network", Operational Research Quarterly, 18, 433-442.

LIEBERHERR K.(1975) Toward Feasible solutions to NP-complete problems Eidgenössische Techn. Hochschule Zurich.

LIN S. (1965) "Computer Solutions of the Traveling Salesman Problem", Bell System Tech.J. 44, 2245-2269.

LIONS J. (1967) :"The Ontario School Scheduling Program". Comp.J. 10, 14-21.

LITTLE J.D.C. (1966) :"The Synchronization of Traffic Signals by Mixed Integer Linear Programming".Opns. Res. 14, 568-594.

LITTLE J.D.C., MURTY K.G., SWEENEY D.W. et KAREL C. (1963) "An Algorithm for the Traveling Salesman Problem", Opns. Res. 11, 979-989.

LORIE J., et SAVAGE L.J. (1955) "Three Problems in Capital Rationing", J. of Bus. 28, 229-239.

LOUKAKIS E. (1972) "Sensitivity Analysis in the Zero-One Integer Linear Programming", Operational Research Technical Report, The University of Manchester.

LOVASZ L. (1968)"On chromatic number of finite set-systems". Acta Mathematica Ac. Sc. of Hungary 19, p.59-64.

LUDOT J.P., DODU J.C. et POUGET J.: "Sur le regroupement optimal des sommets dans un réseau électrique" RIRO, V-1, 17-37.

Mc MAHON G.B. and BURTON P.G. (1967) "Flow Shop Scheduling with the Branch-and-Bound Method". Opns. Res. 15, 473-481.

MANNE A.S. (1960) "On the Job-Shop Scheduling Problem". Opns. Res. 8, 219-223.

——— (1964) "Plant Location under Economies of Scale-Decentralization and Computation". Man. Sci. 11, 213-235.

MAO J.C.T. et WALLINGFORD B.A. (1968) "An Extension of Lawler and Bell's Method of Discrete Optimization with Examples from Capital Budgeting". Man. Sci. 15, B51-B60.

MARANZANA F.E. (1964) "On the Location of Supply Points to Minimize Transport Costs". Operational Research Quarterly 15, 261-270.

MARKOWITZ H.M. et MANNE A.S. (1957) "On the Solution of Discrete Programming Problems". Econometrica 25, 84-110.

MARSTEN R.E. (1971) "An Implicit Enumeration Algorithm for the Set Partitioning Problem with Side Constraints" Ph. D. Dissertation, University of California, Los Angeles.

MARTIN G.T. (1963) "An Accelerated Euclidean Algorithm for Integer Linear Programming" 311-318, in Graves and Wolfe.

——— (1966) "Solving the Traveling Salesman Problem by Integer Linear Programming" CEIT, New York.

MARTIN-LOF A. (1970) "A Branch-and-Bound Algorithm for Determining the Minimal Fleet Size of a Transportation System". Transp. Sci. 4, 159-163.

MASON A.T. et MOODIE C.L. (1971) "A Branch-and-Bound Algorithm for Minimizing Cost in Project Scheduling" Man. Sci. 18, B158-B173.

MIERCORT R.A. et SOLAND R.M. (1971) "Optimal Allocation of Missiles against Area and Point Defenses" Opns. Res. 11, 605-617.

Mac MILLAN C. (1970) Mathematical Programming, Prentice-Hall.

MILLER B.L. (1971) "On Minimizing Non-Separable Functions Defined on the Integers with an Inventory Application" SIAM J. 21, 166-185.

MINOUX M. (1975) Résolution des problèmes de multiflots en nombres entiers dans les grands réseaux. RAIRO , V-3, 21-40.

MITTEN L.G. (1970) "Branch-and-Bound Methods : General Formulation and Properties" Opns. Res. 18, 24-34.

MITRA G.B. Richards, et WOLFENDEN K. (1970) "An Improved Algorithm for the Solution of Linear Programs by the Solution of Associated Di phantine Equations" R.I.R.O.R.I. 47-66.

MORRISON D.R. (1969) "Matching Algorithms" J. Comb. Theory 6, 20-32.

MOTZKIN T. (1966) "Colorings Cocolorings and determinant terms". Theorie des graphes. Journées internationales d'étude. Rome- Dunod, Paris.

MULLER-MERBACH H. (1970) "Approximation Methods for Integer Programming" Johannes Gutenberg University, Mainz, Germany.

——— (1974) Heuristische Verfahren und Entscheidungs- baumverfahren des Operations Research, Handwörterbuch der Betriebswirtschaft (4th ed. E. Grochla, W. Wittmann), Poeschel Verlag, Stuttgart.

——— (1974) : "The role of puzzles in teaching combinatorial programming" in Roy (1975).

MUROGA S. (1970) "Logical Design of Optimal Digital Networks by Integer Programming" pp. 283-348 in J.T. TOU (editor), Advances in Information Systems Science, Volume 3, Plenum Press, New York.

MURTY K.G. (1968) "Solving the Fixed Charge Problem by Ranking the Extreme Points" Opns. Res. 16, 268-279.

MUTH J.F. et THOMPSON G.L. eds. (1963) Industrial Scheduling, Prentice-Hall.

NEMHAUSER G.L. (1966) Introduction to Dynamic Programming, John Wiley & Sons.

——— (1972) "A Generalized Permanent Label Setting Algorithm for the Shortest Path between Specified Nodes" J. Math. Anal. and Appl.

NEMHAUSER G.L. et ULLMAN Z. (1968) "A Note on the Generalized Lagrange Multiplier Solution to an Integer Programming Problem" Opns Res. 16, 450-452.

——— (1969) "Discrete Dynamic Programming and Capital Allocation" Man. Sci. 15, 494-505.

NEMBAUSER G.L. et WIDHELM W.B. (1971) "A Modified Program for Columnar Methods in Mathematical Programming" Opns. Res. 19, 1051-1060.

NEWELL A. et ERNST G. (1965) "The Search for Generality" Proceedings IFIP Congress, Spartan.

NGHIEM PH. T. (1971) "A Flexible Tree-Search Method for Integer Programming Problems" Operations Research, 19, 115-119.

NORMAN R.Z.et RABIN M.O. (1959) "An Algorithm for the Minimum Cover of a Graph" Proc. Amer. Math. Soc. 10, 315-319.

ORCHARD-HAYS W. (1968) Advanced Linear Programming Computing Techniques, Mc Graw-Hill.

ORE O. (1967) The four color problem.Academic Press, New York.

PADBERG M. (1970) "Equivalent Knapsack-type Formulations of Bounded Integer Linear Programs" Man. Sci. Res. Rep. No. 227, Carnegie-Mellon University.

——— (1971) "Simple Zero-One Problems : Set Covering, Matchings and Coverings in Graphs" Man. Sci. Res. Rep. No. 235, Carnegie-Mellon University.

PASTERNAK H. et PASSY U. (1973) "Bicriterion Functions in Annual Activity Planning" 325-341 in M. ROSS, editor, "Operational Research 72" North-Holland, Amsterdam.

PFEFFERKORN C.E. : "A Heuristic problem solving design system for equipement or furniture Layouts" CACM V.18 n°5, 286-297.

PICARD C.F. (1971): Graphes et Questionnaires. tomes 1 et 2. Gauthier-Villars, Paris.

——————— (1969): "Organisation de l'information et pose de problèmes". Revue des questions scientifiques. T 140 n°1, 83-105

PIERCE J.F. (1968) "Application of Combinatorial Programming to a Class of All-Zero-One Integer Programming Problems" Man. Sci. 15, 191-209.

PIERCE J.F. et CROWSTON B. (1971) "Tree Search Algorithms in Quadratic Assignment Problems" Nav. Res. Log. Quart. 18, 1-36.

PIERCE J.F. et LASKY J.S. (1970) "Improved Combinatorial Programming Algorithms for a Class of All-Zero-One Integer Programmin Problems" IBM. Data Processing Division, 428-440.

PIERCE J.F. et LASKY J.S. (1973) All-Zero-One Integer Programming Problems, Management Science 19, n°5 p. 528-543.

PITRAT J. (1966) Réalisation de programmes de démonstration de théorèmes utilisant des méthodes heuristiques. Thèse de Doctorat, Faculté des Sciences, Paris;

——— (1968) Realization of a general game playing program. IFIP. North Holland.

(1970) Un programme général de démonstration de théorèmes. Monographie AFCET, Dunod.

PLA J.M. (1974) :"Confection automatique des roulements programmes d'agents de conduite".Congrès aide à la décision.AFCET Paris.

PLANE D.R. et MILLAN C. (1971) Discrete Optimization : Integer Programming and Network Analysis for Management Decisions, Prentice-Hall.

POLLASTSCHEK M.A. et AVI-ITZHAK B. (1971) "A Class of Deep-Cut Procedures for Integer or Mixed Linear Programs" Rep. No. 85, Faculty of Industrial and Management Engineering, Technion, Israel.

POLYA G. (1962) Mathematical Discovery ; on Understanding, Learning, and Teaching Problem Solving, Wiley, New York.

PUGH R.E. (1972) : "A language for non linear programming problems". Math. Prog. 2, 176-206.

RAO M.R. (1971) "Cluster Analysis and Mathematical Programming" Journal of the American Statistical Association, 66, 622-626.

REITER S. et RICE D.B. (1966) "Discrete Optimizing Solution Procedures for Linear and Nonlinear Integer Programming Problems" Man. Sci. 12, 829-850.

REITER S. et SHERMAN G. (1965) "Discrete Optimizing" SIAM J. 13, 864-899.

REVELLE C., MARKS D. et LIEBMAN J.C. (1970) "An Analysis of Private and Public Sector Location Models" Man. Sci. 11, 692-707.

REVELLE C. et SWAIN R. (1970) "Central Facilities Location" Geographical Analysis, 2, 30-42.

ROBERTS S.D. et VILLA C.E. "On A Multiproduct Assembly Line Balancing Problems" AIIE Trans. 11, 361-364.

ROODMAN G.M. (1972) "Postoptimality Analysis in Zero-One Programming by Implicit Enumeration" Naval Research Logistics Quarterly, 19, 435-447.

ROSENBERG I. (1972) "0-1 Optimization and Nonlinear Programming" Revue Française d'Informatique et de Recherche Opérationnelle, 6, 95-97.

ROSS G.T et SOLAND R.M. (1975) "A branch and bound algorithm for the generaz d assignment problem" Mathematical Programming 8, 91-103.

ROTH R.H. (1969) "Computer Solutions to Minimum Cover Problems" Opns. Res. 17, 455-466.

——— (1970) "An Approach to Solving Linear Discrete Optimization Problems" JACM 17, 303-313.

ROTHSCHILD B. et WHINSTON A.B. (1966a) "On Two-Commodity Network Flows" Opns. Res. 14, 377-387.

——— (1966b) "Feasibility of Two-Commodity Network Flows" Opns. Res. 14, 1121-1129.

ROY B. (1962) "Cheminement et connexité dans les graphes. Application aux problèmes d'ordonnancement" METRA, Série Spéciale n°1.

ROY B. (1969) "Procédures d'exploration par séparation et évaluation (P.S.E.P. et P.S.E.S.), Revue Française d'Informatique et de Recherche Opérationnelle, V-1, 61-90.

——— (1969, 1970) Algèbre Moderne et Théorie des Graphes, Tomes 1 et 2, Dunod, Paris.

——— (1972) "An algorithm for General Constrained Set Covering Problem," in Graph Theory and Computing, Academic Press Inv., p. 267-283.

——— (1975) Combinatorial Programming : Methods and applications.Reidel Pub. by. Dortrecht Holland.

ROY B., BENAYOUN R. et TERGNY J. (1970) "From S.E.P. Procedure to the Mixed Ophelie Program" 419-436, in Abadie.

ROY B., BERTHIER P. et NGHIEM Ph.T. (1965) "Programmes linéaires en nombres entiers et procédure S.E.P." METRA , 4, 441-460.

RUDEANU S. (1969a) "Irredundant Optimization of a Pseudo-Boolean Function" Journal of Optimization Theory and Applications, 4, 253-259.

——— (1969b) "Programmation bivalente à plusieurs fonctions économiques" Revue Française d'Informatique et de Recherche Opérationnelle, 3, 13-30.

——— (1970) "An Axiomatic Approach to Pseudo-Boolean Programming" Mathematik Vestenik, 7, 403-414.

RUBIN D.S. (1970) "On the Unlimited Number of Faces in Integer Hulls of Linear Programs with a Single Constraint" Opns. Res. 18, 940-946.

——— (1972) "Redundant Constraints and Extraneous Variables in Integer Programs" Man. Sci. 18, 423-427.

RUBIN A.A., et HAMMER P.L. (1969) "Quadratic Programming with 0-1 Variables" Rep. No. 53, Faculty of Industrial and Management Engineering. Technion, Istael.

RYSER H.J. (1966) Combinatorial Mathematics.Mathematical association of America.

SAATY T.L. (1970) Optimization in Integers and Related Extremal Problems, Mc Graw-Hill.

SALKIN H.M. (1970) "On the Merit of the Generalized Origin and Restarts in Implicit Enumeration" Opns. Res. 18, 549-554.

. . . (1971) "A Note on Gomory Fractional Cuts" Opns. Res. 19, 1538-1541.

SAMUEL A. (1969) "Some studies in machine learning using the game of checkers" Annual review in automatic programming. Vol. 6 p. 1. Pergamon Press

SANDI S. (1975) "Solution of the machine Loading Problem with binary variables" in Roy 1975.

SCHARLIG A. (1969) "Localisation optimale et théorie des graphes", Cahiers Vilfredo Pareto, 19, Droz, Genève.

SCHECTER M. et HAMMER P.L. (1970) "A Note on the Dynamic Planning of Investment Projects" European Economic Review, 2, 111-121.

SCHRAGE L. (1970) "Solving Resource-Constrained Network Problems by Implicit Enumeration Nonpreventive Case" Opns. Res. 18, 263-278.

SCOTT A.J. (1970) "Location-Allocation Systems : A Review" Geographical Analysis, 2, 95-119.

——— (1971) Combinatorial Programming, Spatial Analysis and Planning, Methuen, London.

SENJU S. et TOYODA Y. (1968) "An Approach to Linear Programming with 0-1 Variables" Man.Sci. 15, B196-B207.

SELBY G.R. (1970) "The use of topolopical methods in computer aided circuit layouts" Ph.D.Thesis London University.

SHAPIRO D. (1966) "Algorithms for the Solution of the Optimal Cost Traveling Salesman Problem" Sc. D. Thesis, Washington University, St. Louis.

SHANNON R.E. et IGNIZIO J.P. (1970) "A Heuristic Programming Algorithm for Warehouse Location" AIIE Trans. 11, 361-364.

SIKLOSSY L. et MARINOV V. (1971) "Heuristic search versus exhaustive search" Second IJCAI, 601-607.

SIKLOSSY L. et DREUSSI J. (1973)"An Efficient Robot Planner which Generates its own Procedures" Third International Joint Conference on Artificial Intelligence, Palo Alto, Californie.

SIKLOSSY L. et ROACH J. (1973)"Proving rhe Impossible is Ippossible is Possible : Disproofs based on Hereditaty Partitions" Third International Joint Conference on Artificial Intelligence, Palo Alto.

SPIELBERG K. (1970) "On Solving Plant Location Problems" 216-234 in E.M.L. BEALE (editor) Applications of Mathematical Programming, English Universities Press, London.

——— (1972) "Minimal Prefered Variables Reduction for Zero-One Programming" IBM Philadelphia Scientific Center, Technical Report n°320-3013.

——— (1973) "A Minimal Inequality Branch-Bound Method" IBM Philadelphia Scientific Center, Technical Report n°320-3024.

SRINIVASAN A.V. (1965) "An Investigation of Some Computational Aspects of Integer Programming" JACM 12, 525-535.

STEINBERG D.I. (1970) "The Fixed Charge Problem" Nav. Res. Log. Quart.17, 217-236.

TABOURIER Y. (1969) Problèmes d'ordonnancement à contraintes purement disjonctives, Revue française de Recherche Opérationnelle V.3.

TAHA H. (1972a) "A Balasian-Based Algorithm for Zero-One Polynomial Programming" Management Science, 18, 328-343.

——— (1972b) "Further Improvements in the Polynomial Zero-One Algorithm" Management Science, 19, 226-227.

TARRY G. (1895) "Le problème des labyrinthes" Nouvelles Annales de Mathématiques, 14.

TEITZ M. et BART P. (1968) "Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph" Operations Research, 16, 955-962.

TERUO S. (1970) "A Method for the Optimal Scheduling of a Project with Resource Constraints" J. Opns. Res. Soc. of Jap. 12, 52-64.

THANGAVELU S.R. et SHETTY C.M. (1971) "Assembly Line Balancing by 0-1 Integer Programming" AIIE Trans. 111, 64-69.

THIRIEZ H. (1969) "Airline Crew Scheduling : a group theoretic approach" Ph. D. Rep. R-67 MIT.

TILLMAN F.H. et LIITSWAGER J.M. (1966) "Integer Programming Formulation of Constrained Reliability Problems" Management Science, 13, 887-899.

TOMLIN J.A. (1970) "Branch and Bound Methods for Integer and Non-Convex Programming" 437-450, in Abadie.

——— (1971) "An Improved Branch-and-Bound Method for Integer Programming" Opns. Res. 19, 1070-1074.

TONGE (1963)" Summary of a heuristic line balancing procedure" Computer and Thought. Mac Graw Hill.

TOREGAS C., SWAIN R., REVELLE C. et BERGMAN L. (1971) "The Location of Emergency Service Facilities" Opns. Res. 19, 1363-1373.

TOYODA Y. (1975)" A Simplified algorithm for obtaining approximate solution to 0-1 programming problems" Mangt Science Vol. 21 n°12.

TRAUTH C.A. et WOOLSEY R.E. (1968) "MESA ; A Heuristic Integer Linear Programming Technique" Res. Rep. SC-RR-68-299, Sandia Labs, Albuquerque, New Mexico.

——— (1969) "Integer Linear Programming : A Study in Computational Efficiency" Man. Sci. 15, 481-493.

TROTTER L.E., Jr., et SHETTY C.M. (1971) "An Algorithm for the Bounded Variable Integer Programming Problem" Dept. of Industrial Engineering, Georgia Institute of Technology.

TUAN N. Ph. (1971) "A Flexible Tree Search Method for Integer Programming Problems" Opns. Res. 19, 115-119.

UNGER V.E. (1970) "Capital Budgeting and Mixed 0-1 Inteter Programming" AIIE Trans. 11, 28-36.

VALENSI E. et CASTAGNE J. (1970) "Problèmes d'affectation. Localisation d'entrepôts" Revue Française d'Informatique et de Recherche Opérationnelle, 3, 31-44.

VON LANZENAUER C.H. (1970) "A Linear Programming Solution to the General Sequencing Problem". CORS Journal, 8, 129-134.

(1971) "A Production Scheduling Model by Bivalent Linear Programming" Management Science, 17, 105-111.

VO KHAC-KHOAN (1969)"La régularisation dans les problèmes combinatoires et son application aux problèmes de tournées et livraisons." RIRO, V-1, 3-21.

————(1972) "Utilisation des coûts pondérés et des variables bivalentes dans le problème de tournées: sectorisation sous contraintes nombreuses" RAIRO V-2,3-19

WAGNER W.H (1968) "An Application of Integer Programming to Legislative Redistricting" presented at the 34th Nat. Meeting of ORSA.

——— (1969) Principles of Operations Research with Applications to Managerial Decisions, Prentice-Hall.

WALKER R.S. (1960) "An Enumerative Technique for a Class of Combinatorial Problems". American Mathematical Society Symposium on Applied Mathe Mathematics Proceedings, 10, 91-94.

WATTERS L.J. (1967) "Reduction of Integer Polynomial Programming Problems to Zero-One Linear Programming Problems" Operations Research, 15, 1171-1174.

WEBB M.H.J. (1971) "Some Methods of Producing Approximate Solutions to Travelling Salesmen Problems with Hundreds or Thousands of Cities" Opnal. Res. Quart. 22, 49-66.

WEINER P. (1975) "Heuristics" Networks 5 p. 101-103.

WEINGARTNER H.M. (1963) Mathematical Programming and the Analysis of Capital Budgeting Problems. Prentice-Hall.

——— (1966) "Capital Budgeting of Interrelated Projects : Survey and Synthesis" Man. Sci. 12, 485-516.

WELSH et POWELL M. (1969)"An upper bound for the chromatic number of a graph"CACM n°12.

WHITE L.J. (1971) "Minimum Covers of Fixed Cardinality in Weighted Graphs" SIAM J. 21, 104-113.

WHITE J.L. et FRANCIS R.L. (1971) "Solving a Segregated Storage Problem Using Branch-and-Bound and Externe Point Ranking Methods" AIIE Trans. III.

WILLIAMS H. (1974) : Experiments in the formulation of integer programming problems. Mathematical Programming Study 2, p. 180.

WITZGALL C. et ZAHN C.T.Jr. (1965) "Modification of Edmonds' Maximum Matching Algorithm" J. Res. Nat. Bur. Stds. 69B, 91-98.

WOLSEY L.A. (1970) "Mixed Integer Programming : Discretization and the Group-Theoretic Approach" Ph. D. Dissertation, Massachusetts Institute of Technology.

WOOD D.C. (1969)"A technique for colouring a graph applicable to large scale timetabling problems" The Computer Journal, Vol. 12, p.317.

WOOLSEY R.E. (1972) "A Candle to Saint Jude, or Four Real World Applications of Integer Programming" Interfaces 2, (Bulletin of TIMS) 20-27.

YOUNG R.D. (1965) "A Primal (All-Integer) Integer Programming Algorithm" J. Res. Nat. Bur. Stds. 69B, 213-250.

——— (1968) "A Simplified Primal (All-Integer) Integer Programming Algorithm" Opns. Res. 16, 750-782.

ZANGWILL W. (1965) "Media Selection by Decisions Programming" Journal of Advertising Research, 5, 30-36.

ZIONTS S. (1968) "On an Algorithm for the Solution of Mixed Integer Programming Problems" Management Science, 15, 113-116.

——— (1969) "Toward a Unifying Theory for Integer Linear Programming" Operations Research, 17, 359-367.

——— (1972) "Generalized Implicit Enumeration Using Bounds on Variables for Solving Linear Programs with Zero-One Variables" Naval Research Logistics Quarterly, 19, 165-181.

TABLE DES SYMBOLES

1. Symboles mathématique usuels.

$\forall$	quel que soit.
$\exists$	il existe au moins un,
{i/}	ensemble des i tels que.
| E |	cardinal de l'ensemble E.
$\in$	appartenant à.
E - F	complémentaire de F dans E.
$\neg$	non logique.
$\lceil a \rceil$	partie entière supérieure de a.
$\lfloor a \rfloor$	" " inférieure de a.
$\overrightarrow{ij}$	arc liant les sommets i et j.

2. Symboles particuliers utilisés au cours du texte.

(Voir aussi le dictionnaire des mots du langage ALICE chapitre I).

$\mathcal{D}$	ensemble des sommets de départ (ensemble source).
$\mathcal{A}$	ensemble des sommets d'arrivée (ensemble but).
f	notation générale d'une fonction de $\mathcal{D}$ vers $\mathcal{A}$.
$\mathcal{E}$	ensemble des images possibles des sommets de $\mathcal{D}$.
K	ensemble des contraintes algébriques d'un problème.
$\mathcal{C}$	contrainte algébrique quelconque.
Z	fonctionnelle des $i \in \mathcal{D}$ à optimiser.
$\underline{Z}$	borne inférieure de Z.
Z_o	valeur de Z pour une solution réalisable obtenue.

TABLE DES MATIERES

INTRODUCTION 1

ENONCE ET RESOLUTION COMPLETE D'UN PROBLEME. 7

I. LE LANGAGE

I.1 - LES PROGRAMMES EXISTANTS. 15

I.2 - LE LANGAGE ALICE : PARTIE DECLARATIVE, OBJETS CHERCHES, CONTRAINTES. 18

I.3 - PRECISION D'UTILISATION DU LANGAGE. DICTIONNAIRE. 21

I.4 - LES CONTRAINTES. 24

I.5 - LES FONCTIONS ECONOMIQUES. 26

I.6 - ECHANTILLON D'ENONCES. 28

II. LES REPRESENTATIONS

II.1 - REPRESENTATION INTERNE SOUS FORME DE GRAPHE. 39

II.2 - IMPLICATIONS GEREES PAR LE GRAPHE INTERNE. 43

II.3 - MANIPULATION FORMELLE D'EXPRESSIONS. 45

II.4 - TRAITEMENT ALGORITHMIQUE DES CONTRAINTES ALGEBRIQUES. 53

II.5 - ORDRE DE SIMPLICITE DES CONTRAINTES. 58

II.6 - TRAITEMENT DES CONTRAINTES DE TROISIEME TYPE ET TROISIEME REPRESENTATION. 60

II.7 - LIAISON ENTRE LE GRAPHE ET LA MANIPULATION FORMELLE. 61

III. LES ARBORESCENCES

III.1 - DESCRIPTION GENERALE DES PROCEDURES ARBORESCENTES. 68

III.2 - SOLUTIONS REALISABLES ET DISTANCE A L'OPTIMUM. 69

III.3 - TROIS TYPES D'ENUMERATION. 72

III.4 - LES DIFFERENTS PRINCIPES DE SEPARATION. 73

III.5 - CRITERES DE CHOIX. 77

III.6 - ORDRE DE GESTION DES CRITERES. 79

III.7 - PREMIERE PHASE : RECHERCHE D'UNE SOLUTION REALISABLE. 81

III.8 - DEROULEMENT DU PROGRAMME DANS LES PHASES 2 ET 3. 87

III.9 - RESOLUTION DES EXEMPLES DE LA PARTIE I . 92

IV. LES RESULTATS

TABLE DES RESULTATS. 109

IV.1 - LES PUZZLES. 112

IV.2 - PROGRAMMATION MATHEMATIQUE EN NOMBRES ENTIERS. 135

IV.3 - PROBLEMES D'ORDONNANCEMENT AVEC DATES LIMITES, DECOUPE ET BIN PACKING. 151

IV.4 - PROBLEMES D'ORDONNANCEMENTS DISJONCTIFS. 154

IV.5 - PROBLEMES DE DECOUPE ET AIDE A LA CONCEPTION. 155

IV.6 - PROBLEMES DU VOYAGEUR DE COMMERCE ET PROBLEMES VOISINS. 164

IV.7 - PROBLEMES DE COLORATION ET D'EMPLOI DU TEMPS. 170

IV.8 - PROBLEMES DE FABRICATION ET LOCALISATION. 175

IV.9 - LE PROBLEME DES ROULEMENTS A LA S.N.C.F. 178

V. REFLEXIONS ET CONCLUSION 185

BIBLIOGRAPHIE

TABLE DES SYMBOLES